MASTER EN SCIENCES INFORMATIQUES

Introduction à X.500 : principes et définitions abstraites

Brunet, Maxime

Award date:
1992

Awarding institution:
Universite de Namur

Facultés Universitaires Notre-Dame de la Paix
Institut d'Informatique

Rue Grandgagnage, 21, B-5000 Namur (Belgium)

Introduction à X.500 : principes et définitions abstraites

par

Brunet Maxime

Promoteur : Professeur Philippe van BASTELAER

Mémoire présenté en vue de l'obtention
du titre de Licencié et Maître en Informatique

Année académique 1991-1992

Remerciements

Nous avons particulièrement à coeur de remercier l'ensemble des personnes qui nous ont aidés lors de la rédaction de ce travail.

Nos remerciements s'adressent à :

Monsieur Philippe van Bastelaer pour l'ensemble de ces conseils, pour le soutien et pour l'attention qu'il a manifestée durant la rédaction de ce travail;

Monsieur John Berge pour nous avoir permis d'effectuer notre stage chez SITPRO et pour la documentation qu'il nous a fournie;

Monsieur et Madame Brunet-Houttaeve pour nous avoir offert la possibilité de faire ces études.

Résumé :

La première partie de ce mémoire expose les recommandations X.500 datant de 1988 ; la deuxième, celles datant de 1992. Pour chacune de celles-ci, nous approchons les principes de fonctionnement du Directory (contrôle des accès, sécurité, protocoles, traitement de l'information...) ainsi que les définitions abstraites des éléments du Directory. Quelques exemples de requêtes sont aussi proposés afin d'exposer, dans sa globalité le fonctionnement et l'implémentation abstraite du Directory.

Abstract :

The first part of this survey exposes the X.500 recommendations from 1988 ; the second one, those from 1992. For each of them, we approach the Directory functioning principles (access control, security, protocols, information processing...) and also the Directory elements abstract definitions. Some request examples are proposed to show, in its entirety the Directory functioning and abstract implementation.

Table des matières.

Chapitre 0 : Introduction

Partie 1 : Revue des recommandations X.500 (1988)

Chapitre 1

Chapitre 2

Partie 2 : Revue des recommandations X.500 (1992)

Chapitre 3

Chapitre 4 : Modèle de sécurité

Chapitre 5: Exemple de contrôle d'accès de base

Chapitre 6 : Le service abstrait du directory

Chapitre 7

Chapitre 8 : La nouvelle recommandation X.5rp

Chapitre 9 : Exemple de modélisation des connaissances

0.1 Introduction.

Ce chapitre, qui sert d'introduction à ce mémoire, a pour but d'introduire l'utilité de X.500, avant de passer lors des chapitres suivants à un exposé détaillé des fonctions et de l'implémentation abstraites du Directory.

0.2 Buts du Directory.

Le Directory a été créé dans un but bien précis : la "gestion facile" des adresses électroniques de réseaux, de modems... On a par la suite élargit l'utilisation du Directory à tout ce qui pouvait se trouver dans un répertoire général comme par exemple des renseignements sur tous les employés d'une firme dont on détenait déjà l'adresse physique...

0.3 Impacts du Directory dans le contexte des communications.

0.3.1 Impacts sur les noms et l'adressage.

L'impact principal du Directory est qu'il permet d'installer une communication entre deux partenaires, sans que ceux-ci connaissent la localisation et l'adresse du réseau, de la machine de leur correspondant.

Afin d'établir un dialogue, le modèle OSI demande préalablement l'établissement d'une connexion, et ceci peu importe l'application employée. L'appelant doit fournir le nom de l'entité application qu'il désire contacter au protocol de connexion. Ce nom sera ensuite transformé en adresse et information de routage, grâce à des tables de transformation contenant cette information (ces tables étant gérée individuellement).

Mais maintenant, avec l'arrivée du Directory, ces tables deviennent inutiles. En effet, le Directory offre la facilité suivante : on lui communique le nom de celui qu'on désire joindre et il renvoie les différents sélecteurs servant au routage interne et l'adresse réseau par exemple.

Bien sûr, on pourrait se dire que tout ne marchait pas si mal avec les tables de transformation. Oui, mais que se passerait-il si votre interlocuteur avait déménagé? Vous devez gérer ces tables vous mêmes. Maintenant le Directory le fait pour vous...

En ce qui concerne les noms, le Directory permet l'utilisation de ce qu'on pourrait appeler une dénomination facile pour l'utilisateur (user friendly naming). Cela permet de référencier des objets au moyen de noms faciles à retenir pour l'utilisateur, ou en tous cas moins complexes que ceux employés par un ordinateur.

0.3.2 Impacts sur le processus d'authentification.

Beaucoup d'applications comme par exemple FTAM, utilisent des paramètres convoyés par le A-Associate pendant l'ouverture de la communication afin de pourvoir aux services de contrôle d'accès et d'authentification. Ce contrôle peut être remplacé par un appel au Directory afin de savoir par exemple si le correspondant a les privilèges nécessaires à l'ouverture de tel ou tel fichier, à la modification d'une base de données. Ceci est rendu possible par le Directory, car on peut y stocker tous ses privilèges pour toutes les applications qu'il peut appeler.

Le processus d'authentification peut être résolu de la même façon. En effet, un partenaire peut vérifier si son correspondant est bien qui il dit être en utilisant la clef publique de celui-ci, contenue dans son entrée de Directory et sa propre clef privée.

0.3.3 Impacts sur les possibilités d'un système.

Grâce au Directory, il est aussi possible de savoir si un système fournit tel ou tel service. Ce service du Directory est possible en ajoutant cette information dans l'entrée correspondante à ce système.

0.3.4 Conclusion.

Nous venons d'énoncer ici trois impacts possibles. Cependant, il faut bien se mettre en tête que le Directory peut fournir d'énormes services pour autant qu'il contienne l'information requise pour ces services. Pour ainsi dire, le Directory peut tout. Par exemple, on pourrait imaginer une entrée pour chaque habitant de la planète, contenant un certain nombre d'informations, comme le numéro de la plaque minéralogique de la voiture possédée par cette personne....

En fait le Directory n'est ni plus ni moins qu'une énorme base de données contenant dans un premier temps l'information nécessaire aux processus d'authentification, de résolution des noms.... sans l'obligation d'une gestion individuelle interne. Ceci est d'ailleurs très utile lors des modifications car vous n'avez pas à prévenir tous vos partenaires de cette modification (par exemple, un numéro de téléphone). La modification se fait au sein du Directory et uniquement là. Par la suite, le Directory pourrait contenir n'importe quelle information.

0.4 Exemple : le cas de l'EDI.

Comme nous l'avons déjà dit, le Directory peut contenir n'importe quelle information. C'est d'ailleurs pourquoi les utilisateurs d'EDI aimeraient y trouver l'information utile à leurs échanges.

Prenons par exemple le cas d'une entreprise X désirant connaître le prix d'un produit Y. Supposons aussi que cette entreprise utilise l'EDI lors de ces transactions commerciales. En admettant que le Directory contienne toute l'information nécessaire (que nous détaillerons par la suite), celui-ci peut être d'une utilité capitale. Que va-t'il se passer ?

L'entreprise X est à la recherche du produit Y au meilleur prix. Tout d'abord elle envoie une requête au Directory lui demandant les noms des entreprises fournissant ce produit Y. Le Directory renvoie cette liste. De cette liste, l'entreprise retire les fournisseurs trop éloignés et ceux n'utilisant pas l'EDI (notons que ceci peut-être demandé dans la requête). Ensuite, l'entreprise contacte les fournisseurs en utilisant EDI et le Directory pour les adresses physiques des fournisseurs. Le message envoyé est "request for quotes". L'entreprise reçoit ainsi les offres de prix des fournisseurs.

Le Directory doit donc contenir de l'information supplémentaire pour répondre à cette requête. Cette information concerne l'EDI. Lors de l'utilisation du Directory par EDI, cette information est généralement, outre l'adresse et les informations de routage, l'ensemble des versions, des éléments de donnée, des messages, des jeux de caractères, des codes supportés.

On voit, dès lors, rien que sur ce petit exemple, l'utilité de l'emploi du Directory en EDI car il peut permettre la communication en EDI de deux partenaires n'ayant pas d'accord d'interchange préalable. En fait lors de cet accord, on communique à la partie adverse les renseignements contenus maintenant dans le Directory.

1.0. Introduction.

L'ensemble de ce chapitre a pour objet le Directory X.500. Dans ce qui suit, on trouvera une revue détaillée des recommandations du CCITT portant sur le Directory. Celles-ci ont été publiées sous le nom de code X.500 en 1988. Une révision de ces recommandations devraient être publiée dans le courant de l'année 1992. Nous tâcherons de présenter l'évolution des recommandations avec l'aide de cette nouvelle publication lors de chapitres suivants.

La troisième partie de ce chapitre parlera de l'utilité du Directory X.500 et s'attardera plus longuement sur les impacts de ce Directory sur le modèle ISO.

On peut trouver la définition formelle en ASN.1 de tous les éléments définis dans la suite de ce chapitre en annexe [X.500.1]. Une explication de la syntaxe ASN.1 est également incluse en annexe [X.500.2].

1.1. Revue des Recommandations X.500 (1988).

1.1.0. Introduction et Définitions Générales. [X.500]

Le Directory est une collection de systèmes "ouverts" (Open Systems) coopérant les uns avec les autres afin de pourvoir aux services offerts par ce Directory.

Le Directory offre divers services requis par les applications du modèle ISO, par les autres entités de ISO et par les services de télécommunication. Parmi tous ces services, on trouve par exemple :

- des services permettant une dénomination des objets plus facile pour l'utilisateur ("User-Friendly Naming"). Ceci offre la possibilité de référencier les objets d'une manière plus aisément assimilable et utilisable par des utilisateurs humains.
- des services procurant un adressage par nom ("Name-to-Address Mapping"). Cette façon de procéder crée un lien direct entre les objets et leur localisation.

Avant de passer à une revue plus détaillée du Directory, nous pensons qu'il est préférable de définir deux éléments importants :

- un utilisateur du Directory (Directory User) est une entité ou une personne qui accède au Directory.
- la base de données du Directory (Directory Information Base) est un ensemble d'informations administrées par le Directory. Cet élément est dénoté sous le nom DIB.

1.1.1. Compte-Rendu Général du Directory. [X.500]

1.1.1.1. Le Directory et ses Utilisateurs. [X.500] [X.501]

Comme défini ci-dessus, le Directory est une collection de systèmes "ouverts" qui coopèrent les uns avec les autres afin d'administrer une base de données contenant des informations sur des objets du "monde réel".

Les utilisateurs du Directory, càd des personnes et des programmes, peuvent lire ou modifier l'information ou une partie de celle-ci, contenue dans le DIB, si ils en ont l'autorisation. Chaque utilisateur du Directory est représenté lors de l'accès au Directory par un Directory User Agent (DUA) qui est considéré comme étant un processus-application. Un DUA accède réellement au Directory et interagit avec lui dans le but d'obtenir des services demandés par un utilisateur particulier. Ces concepts sont représentés par la figure suivante :

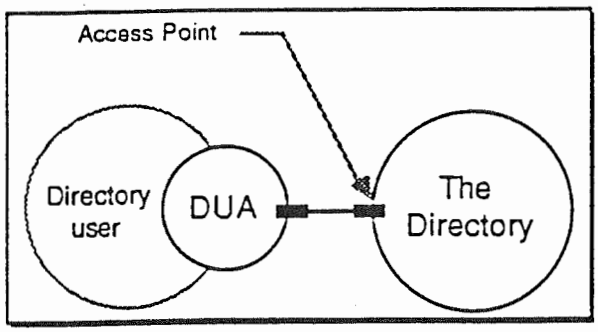


Fig 1.1. Accès au Directory.

Chaque Directory User Agent (DUA) représente précisément un utilisateur du Directory. Tout accès au Directory se fait par un point d'accès (Access Point).

L'information contenue dans le Directory est communément appelée Directory Information Base (DIB). La structure du DIB est le sujet du paragraphe 1.2. de ce chapitre.

Le Directory procure à ses utilisateurs un ensemble bien défini de moyens d'accès appelé le service abstrait (Abstract Service) du Directory. Ce service, qui est décrit au paragraphe 1.4. de ce chapitre, offre de simples capacités de modification et de récupération des données.

1.1.1.2. Modèle fonctionnel du Directory. [X.501]

Le Directory est composé d'un ensemble contenant un ou plusieurs processus-application dénommés Directory System Agents (DSAs). Chacun de ces DSAs offre zero, un ou plusieurs points d'accès au Directory. Ceci est illustré par la figure 1.2.

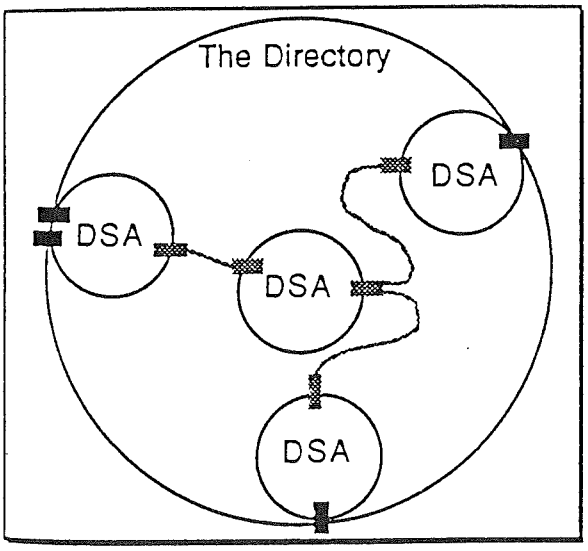


Fig 1.2. Un Directory composé de plusieurs DSAs.

Quand un Directory est composé de plus d'un Directory System Agent (DSA), il est dit distribué. Il est très probable que le Directory sera distribué, voire même mondialement distribué. Considérant cette éventualité, des modèles de Directory ont été développés et sont détaillés dans le paragraphe 1.5. de ce chapitre.

Une paire de processus-application, soit un DUA et un DSA, soit deux DSAs, qui ont besoin d'interagir afin d'obtenir un service du Directory peuvent être situés dans deux systèmes différents. Une telle interaction est effectuée avec l'aide de protocoles ISO du Directory qui sont définis au paragraphe 1.6. de ce chapitre.

1.1.1.3. Modèle Organisationnel du Directory. [X.500] [X.501]

Un ensemble composé d'un ou plusieurs DSAs et de zéro ou plusieurs DUAs est administré par une organisation unique. Un tel ensemble est appelé un Directory Management Domain (DMD).

Un Directory Management Domain peut être géré par une administration (ADDMD) ou par une organisation privée (PRDMD).

Chaque Directory System Agent (DSA) est administré par une autorité administrative (Administrative Authority). Cette entité a le contrôle sur toutes les entrées contenues dans ce DSA. Ceci comprend aussi des responsabilités envers le schéma du Directory (Directory Schema) employé pour réguler la création et la modification des entrées. Le directory schéma est l'objet du paragraphe 1.3. de ce chapitre.

Il existe aussi une administration responsable de la structure et de l'allocation des noms. Elle se nomme Naming Authority. Le rôle de l'autorité administrative est d'implémenter la structure de ces noms dans le schéma du Directory.

1.1.1.4. Le Modèle de Sécurité.

Le Directory appartient à un environnement où plusieurs autorités offrent l'accès à leurs fragments du DIB. De tels accès doivent être effectués en accord avec des règles de securité.
Il existe deux types de règles de sécurité au sein du Directory. Celles-ci définissent la politique d'autorisation et la politique d'authentification.

La politique d'authenfication est le sujet de la section 1.1.7.

La politique d'autorisation ne sera pas traitée dans ce chapitre.

1.1.2. Le DIB - Directory Information Base. [X.501]

Le Directory Information Base est composé d'information concernant divers objets. Le DIB contient un certain nombre d'entrées (Directory Entries), chacune d'elles consistant en une série d'information au sujet d'un objet. Chaque entrée est composée d'attributs ayant un type et une ou plusieurs valeurs. La présence de tel ou tel type d'attribut dépend de la classe d'objet décrite par cette entrée.

Tous les éléments employés ci-dessus sont définis dans cette section.

1.1.2.1. Les Objets et Classes d'Objets. [X.501]

Le but du Directory est de contenir et de permettre des accès à des informations concernant des objets particuliers qui existent dans un certain "monde".

On définit un objet comme étant quelque chose d'identifiable, qui porte un nom dans ce "monde".

Le "monde" en question est généralement celui des télécommunications et du traitement de l'information.

Une classe d'objets identifie une famille d'objets qui partagent certaines caractéristiques. Chaque objet appartient au moins à une classe d'objets. Une classe d'objets peut être une sous-classe ou une super-classe d'une autre classe d'objets.

Quand un objet appartient à une sous-classe, il partage les caractéristiques des objets de sa classe mais aussi celles de sa super-classe. On peut avoir des sous-classes de sous-classe, etc, et cela jusqu'à une profondeur arbitraire.

1.1.2.2. Les Entrées du Directory. [X.501]

Les entrées du Directory constituent le Directory Information Base (DIB) et contiennent des informations au sujet d'un seul objet. Pour chacun de ces objets, il y a précisément une seule entrée au Directory. Cette entrée constitue la première information dans le DIB concernant cet objet.

Pour chaque objet particulier, on peut avoir en addition de son entrée "réelle", une ou plusieurs entrées alias (Alias Entries) qui sont utilisées pour procurer des noms alternatifs à cet objet.

Chaque entrée contient une indication sur la classe d'objets et les super-classes de cette classe d'objets à laquelle cette entrée est associée. La classe d'objets d'une entrée alias est alias.

1.1.2.2.1. Composition d'une Entrée. [X.501]

La figure suivante décrit la composition d'une entrée dans le Directory Information Base (DIB).

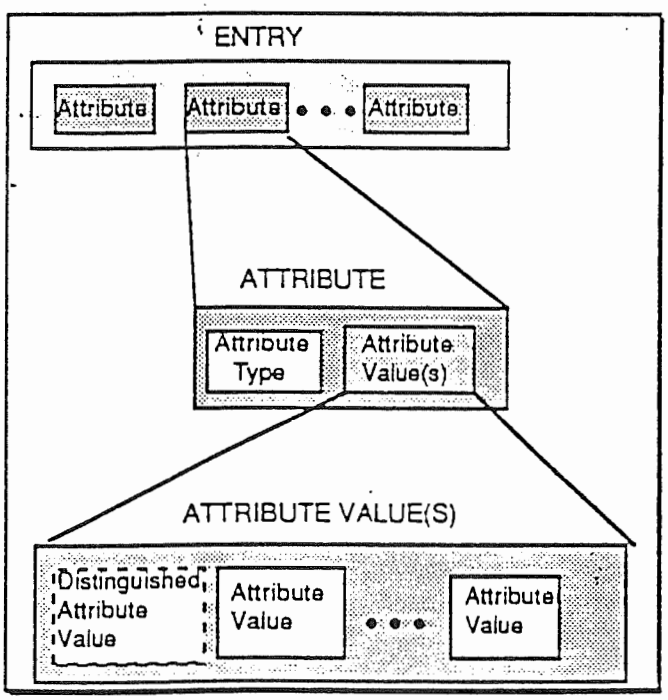


Fig 1.3. Composition d'une entrée.

Chaque entrée consiste en un ensemble d'attributs. Chaque attribut fournit une partie de l'information, ou décrit une caractéristique particulière de l'objet correspondant à cette entrée. Une entrée peut par exemple contenir une information sur le nom de l'objet et sur son adresse.

Un attribut est constitué par :
- un type d'attribut qui identifie la classe d'information fournie par l'attribut.
- une ou des valeurs d'attribut qui sont une instance particulière de la classe d'information apparaissant dans l'entrée.

1.1.2.2.2. Types d'attribut. [X.501]

Certains types d'attribut seront définis suivant un standard international, d'autres par une autorité administrative nationale ou par une organisation privée. Ceci implique donc qu'un certain nombre d'autorités seront responsables de l'assignation des types d'une manière assurant l'unicité de chacun des types. Ceci est résolu par l'identification de chaque type d'attribut à un identifiant d'objet (Object Identifier) lors de la définition de ce type.

On remarquera que tous les attributs d'une entrée doivent être de types différents.

Il existe bon nombre de types d'attribut connus par le Directory et qu'il utilise lui-aussi comme par exemple :

- ObjectClass : un attribut de ce type apparaît dans chaque entrée et indique la classe d'objet à laquelle appartient l'entrée.
- AliasedObjectName : un attribut de ce type est inclus dans chaque entrée de type alias et contient le nom distingué (voir 1.1.2.3.) de l'objet que l'entrée de type alias décrit.

1.1.2.3. Les Noms. [X.501]

Un nom (Directory Name) est une construction qui identifie un objet particulier parmi un ensemble d'objets. Un nom doit être non-ambigu pour n'identifier qu'un seul objet. Un nom ne doit pas cependant être unique dans le sens ou plusieurs noms peuvent dénoter le même objet.

Syntaxiquement, chaque nom est une séquence ordonnée de noms distingués relatifs (Relative Distinguished Name) définis au paragraphe 1.2.3.1. de ce chapitre.

Le nom donné à la racine du DIB est la séquence vide. Chaque sous-séquence initiale du nom d'un objet est aussi le nom d'un autre objet. La collection d'objets ainsi identifiés commence avec la racine du DIB et se termine avec l'objet référencié par ce nom.

1.1.2.3.1.Noms Distingués Relatifs - Relative Distinguished Names

Un RDN consiste en un ensemble d'attributs et leurs valeurs. Les RDNs de toutes les entrées ayant le même supérieur sont distincts. Il y va de la responsabilité de l'autorité attribuant les noms (Naming Authority) d'assurer une assignation appropriée des valeurs d'attributs distingués.

1.1.2.3.2 Noms Distingués - Distinguished Names. [X.501]

Le nom distingué d'un objet donné est défini comme étant la séquence de RDNs de l'entrée représentant l'objet associée avec celle de toutes les entrées supérieures. Cette façon d'agir suffit pour référencer un objet.

Le nom distingué d'une entrée de type alias est formé de la même façon.

La figure suivante illustre le concept des RDNs et des noms distingués.

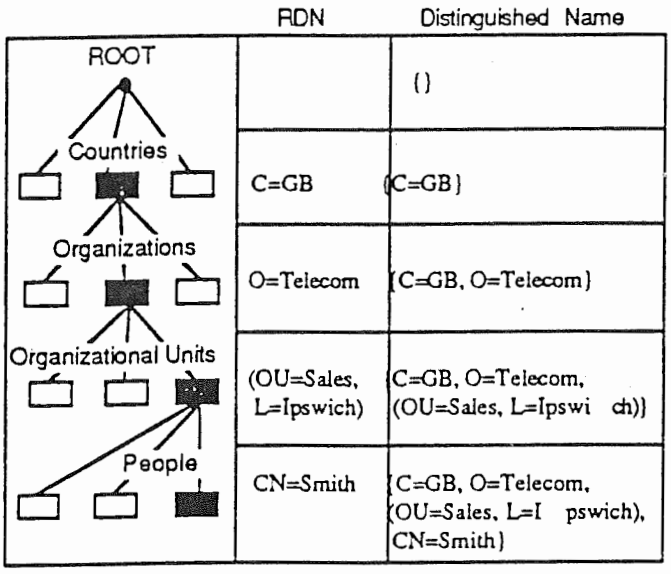


Fig 1.4. Détermination des noms distingués.

1.1.2.3.3. Alias. [X.501]

Les alias permettent aux entrées représentant des objets d'avoir plusieurs supérieurs immédiats. C'est de cette façon que les alias fournissent les bases pour l'utilisation de noms alternatifs.

Un objet peut avoir un ou plusieurs alias. Il s'en suit que plusieurs entrées alias peuvent pointer vers la même entrée. Un alias ne peut avoir d'alias ni d'entrée subordonnée. Les entrées de type alias sont toujours des feuilles du DIB.

Le nom de l'entrée correspondant à l'alias est indiquée dans celle-ci par un attribut. C'est ainsi que le Directory peut retrouver l'entrée référencée par l'alias.

1.1.3. Le schéma du Directory - Directory Schema. [X.501]

Le schéma du Directory est un ensemble de définitions et de contraintes concernant :

- la structure du DIT (voir paragraphe 1.3.1. de ce chapitre).
- les différentes manières de nommer les entrées.
- la façon de stocker l'information dans une entrée.
- les attributs utilisés pour représenter l'information.

Formellement, le schéma du Directory comprend un ensemble de :

- définitions de structure du DIT qui spécifient les noms distingués que les entrées peuvent avoir et la façon dont elles sont en relations les unes vis-à-vis des autres au sein du DIT.

- définitions de classes d'objets qui spécifient l'ensemble des attributs obligatoires ou conditionnels qui doivent ou qui peuvent se trouver au sein d'une entrée de cette classe d'objets.

- définitions des types d'attribut qui spécifient l'identifiant d'objet (voir 1.1.2.2.2.) par lequel est connu un attribut, sa syntaxe et s' il peut avoir des valeurs multiples.

- définitions de syntaxe des attributs qui spécifie pour chacun d'eux sa représentation en ASN.1

La figure suivante résume les relations entre ,d'un côté, les définitions du schéma et de l'autre, le DIT, les entrées du Directory, les attributs et leur(s) valeur(s).

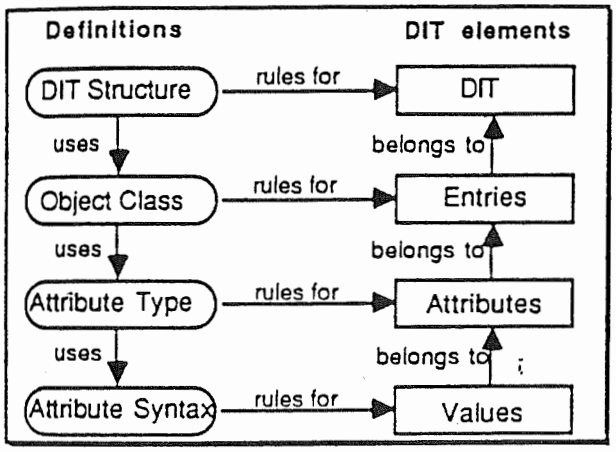


Fig 1.5. Vue générale du schéma du Directory.

Dans le cas où le DIB est distribué, le schéma du Directory l'est aussi. Chaque autorité administrative régule la part de schéma du Directory correspondant à sa part de DIB.

1.1.3.1. Le DIT - Directory Information Tree. [X.500] [X.501]

Les entrées du Directory Information Base sont arrangées selon la forme d'un arbre appelé le Directory Information Tree (DIT). Dans ce DIT, les sommets représentent les entrées. On notera que les entrées près de la racine de l'arbre représentent souvent des objets tels que des pays ou des organisations tandis que les entrées situées près des feuilles de l'arbre représentent des applications ou des personnes.

La figure 1.6. représente un hypothétique Directory Information Tree (DIT).

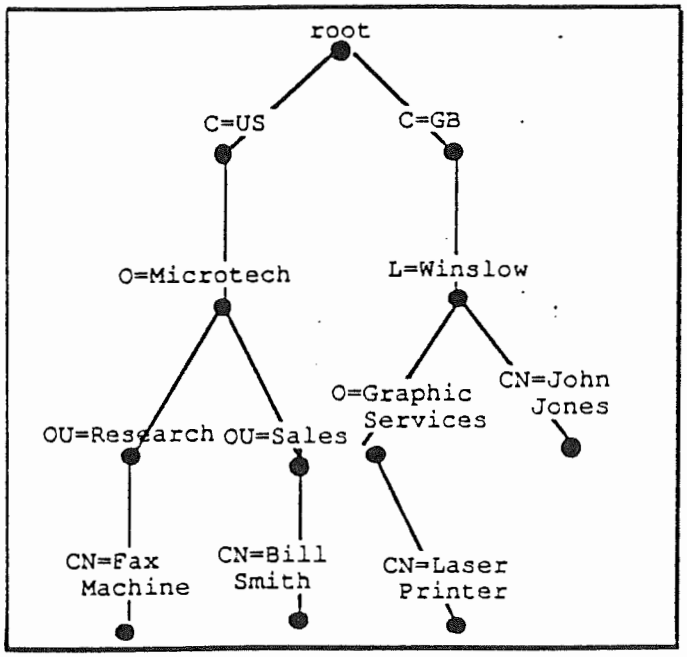


Fig 1.6. Un exemple de DIT.

1.1.4. Le Service Abstrait du Directory - Directory Abstract Service. [X.511] [X.518] [1.407]

1.1.4.0. Introduction.

Cette section propose une revue des services offerts par le Directory à ses utilisateurs représentés par leurs DUAs. Tous les services procurés par le Directory le sont en réponse à des requêtes émises par les DUAs. Différents types de requêtes sont offerts :

- des requêtes permettant l'interrogation du Directory. Elles sont décrites aux paragraphes 1.4.2. et 1.4.3. de ce chapitre.
- des requêtes destinées à la modification du Directory. Elles sont détaillées au paragraphe 1.4.4. de ce chapitre.

Le Directory émet lors de chaque requête un rapport sur le résultat de celle-ci. Les résultats anormaux sont dus à des erreurs. Celles-ci sont expliquées au paragraphe 1.4.5. de ce chapitre.

Il faut signaler que le Directory assure son integrité lors de requêtes modifiant le Directory Information Base (DIB). Il doit toujours obéir aux règles du schéma du Directory.

Comme déjà dit précédemment, les services du Directory sont procurés à travers des points d'accès (Access Points) aux DUAs agissant sous l'égide de leurs utilisateurs. En principe, ces points d'accès peuvent être de différents types et donc procurer différents services.

Il est préférable de considérer le Directory comme un objet ayant un certain nombre de "ports" de types différents. Chaque port définit une interaction particulière du Directory avec un DUA. Chaque point d'accès correspond alors à une combinaison particulière de types de port. Il existe trois types de port : le readport, le searchport et le modifyport.

1.1.4.1. Opérations de Connexion et de Déconnexion - Bind and Unbind Operations. [X.511]

Les opérations de connexion (Directory Bind) et de déconnexion (Directory Unbind) sont utilisées respectivement au début et à la fin d'une période particulière d'accès au Directory.

1.1.4.1.1. Directory Bind Operation. [X.511]

Lors d'une connexion au Directory, un DUA doit indiquer par quel type de port il désire passer. Ce choix est en relation directe avec le type de requête pour laquelle on se connecte au Directory.

Les arguments passés lors de la connexion sont les suivants :

- l'identité de l'utilisateur (Credentials). Pour plus de renseignements voir 1.1.7.
- la ou les versions de service requis par le DUA.

Si la procédure de connexion réussit, un résultat est retourné au DSA.

- l'identité du DSA auquel le DUA s'est connecté.
- la ou les versions de service offerts par de DSA.

En cas d'échec lors de la tentative de connexion, une erreur est retournée au DUA demandeur. Cette erreur a pour cause :

- la version du service requis n'est pas supportée par le DSA. Dans ce cas, le DSA retourne la ou les versions de requête qu'il peut supporter.
- un accès est refusé pour cause d'authentification inappropriée.

1.1.4.1.2. Directory Unbind Operation. [X.511]

Cette opération est utilisée pour se déconnecter d'un certain port d'un DSA à la fin d'une période d'accès.

1.1.4.2. Opérations de Lecture - Directory Read Operations.

Il existe en fait deux types d'opérations de lecture : l'opération read et l'opération compare. Cependant l'opération abandon est groupée avec les opérations de lecture pour une raison de facilité.

1.1.4.2.1. Read Operation. [X.511]

Une opération de lecture (Read) est utilisée pour extraire des informations concernant une entrée du DIB explicitement identifiée. Elle peut aussi servir à vérifier un nom distingué (Distinguished Name).

Cette opération a comme paramètre un ensemble d'arguments comprenant chaque fois :

- un nom identifiant l'entrée dont on demande l'information. Ce nom peut être un alias.
- une sélection indiquant quelle partie d'information contenue dans cette entrée est requise.

Dans le cas où l'opération réussit, les informations demandées sont renvoyées. Dans le cas contraire, une erreur est signalée. Cette erreur peut être due par exemple à un nom ou un attribut inconnu, à un abandon ou à une violation de sécurité...

1.1.4.2.2. Compare Operation. [X.511]

Une opération de comparaison (Compare) est utilisée afin de comparer une ou des valeurs passées comme paramètres à une ou plusieurs valeurs d'un type d'attribut d'une entrée particulière du DIB.

Les paramètres de cette opération sont :

- un nom qui identifie une entrée particulière du DIB.
- la valeur et le type d'attribut à comparer.

Si l'opération se déroule sans problème, le résultat contient les éléments suivants :

- un nom distingué si le nom passé en paramètre identifiait une entrée de type alias.
- le résultat de la comparaison (True or False).
- un booléen indiquant si la comparaison a été effectuée sur l'entrée originale ou sur une copie.

Si une erreur s'est produite, le DUA en est informé.

1.1.4.2.3. Abandon Operation. [X.511]

Les opérations interrogeant le Directory peuvent être interrompues par une opération d'abandon si l'utilisateur n'est plus intéressé par le résultat de la requête.

Cette opération a un seul paramètre identifiant l'opération à interrompre. Si tout se passe sans problème, l'utilisateur est averti et l'opération interrompue se termine par une erreur du type abandon. Si il n'est pas possible d'interrompre l'opération en cours (parce qu'elle n'existe pas par exemple), un message d'erreur est envoyé à l'utilisateur.

1.1.4.3. Opérations de Recherche - Directory Search Operations.

Les opérations de recherche sont au nombre de deux et sont décrites ci-dessous.

1.1.4.3.1. List Operation. [X.511]

Une opération d'énumération (List Operation) est utilisée pour obtenir une liste des subordonnés immédiats d'une entrée explicitement définie. Dans certains cas, la liste résultat peut être incomplète.

Cette opération a pour argument un ensemble de paramètres contenant chaque fois un nom identifiant de l'entrée dont on désire connaître les subordonnés immédiats.

Quand l'opération réussit, et ceci peu importe le fait que l'entrée aie ou n'aie pas de subordonnés immédiats, un résultat est retourné au demandeur. Ce résultat contient les éléments suivants :

- le nom distingué de l'entrée en question dont on désirait connaître les subordonnés immédiats dans le cas où le nom passé en paramètre était un alias.
- pour chaque subordonné immédiat on trouve, le nom distingué relatif (RDN) de cette entrée, une information indiquant si l'entrée subordonnée est un alias et une information stipulant s'il s'agit d'une entrée réelle ou d'une copie. On notera que les alias trouvés ne sont pas deréférencés.
- lorsque la liste résultat est incomplète :
- un élément indiquant qu'une limite temporelle ou de taille ou administrative a été dépassée.
- un ensemble de noms distingués d'objets et de points d'accès permettant au DUA de continuer l'énumération en contactant ces autres points d'accès si des parties du DIT n'ont pas été explorées.
- un paramètre indiquant l'impossibilité de fournir une liste complète.

L'opération retourne une erreur quand le nom passé en paramètre est faux, en cas d'abandon, en cas de violation de privilèges...

1.1.4.3.2. Search Operation. [X.511]

Une opération de recherche est utilisée pour retrouver dans une partie du DIT des entrées d'un intéret particulier et de retourner une selection d'informations concernant cette entrée.

Les arguments de cette opération sont :

- un nom identifiant une entrée à partir de laquelle la recherche doit prendre place (nom de base).
- la profondeur de la recherche : l'entrée identifiée ou les subordonnés immédiats ou tous les subordonnés de cette entrée.
- un filtre qui est utilisé pour éliminer les entrées inadéquates.
- un booléen indiquant s'il faut déréférencer les alias. Dans l'affirmative, la recherche continuera dans le sous-arbre dont la racine est l'entrée pointée par l'alias.
- un ensemble de types d'attribut dont la valeur doit être prise en compte.

L'opération réussit dans le cas ou l'entrée correspondant au nom de base a pu être localisée et ceci peu importe le contenu de la recherche. Le résultat contient un ensemble de paramètres consistant, pour chaque entrée valide, en :

- le nom distingué de l'entrée identifiée par le nom de base si celui-ci était un alias (ceci n'apparait qu'une fois).
- les informations demandées.
- lorsque le résultat est incomplet :

- un élément indiquant qu'une limite temporelle ou de taille ou administrative a été depassée.
- un ensemble de noms distingués d'objets et de points d'accès permettant au DUA de continuer la recherche en contactant ces autres points d'accès si des parties du DIT n'ont pas été explorées.
- un paramètre indiquant l'impossibilité de fournir un resultat complet.

Si l'opération échoue, une erreur est reportée (Du même genre que celles de l'opération d'énumeration).

1.1.4.4. Opérations de Modification - Directory Modify Operations. [X.511]

Il existe quatre opérations de modification du Directory : AddEntry - RemoveEntry - ModifyEntry - ModifyRDN qui sont définies dans ce qui suit.

1.1.4.4.1. Ajout d'une Entrée - Add Entry. [X.511]

Cette opération permet d'ajouter au DIT une entrée de type objet ou alias qui est toujours une feuille du DIT. Elle a pour arguments les paramètres suivants :

- le nom distingué de l'entrée à ajouter. On remarquera que ce nom permet d'identifier le supérieur immédiat de cette entrée, en lui retirant le dernier nom distingué relatif(RDN). Pour que l'opération réussisse, ce supérieur immédiat doit exister.
- un ensemble de types et de valeurs d'attribut constituant l'information à stocker dans l'entrée. On notera que l'ajout d'une nouvelle entrée doit respecter le schéma du Directory. Cependant quand l'entrée créée est un alias, aucune précaution n'est prise pour vérifier que l'entrée pointée est valide.

Si l'opération réussit, l'utilisateur en est averti. Dans le cas contraire, un message d'erreur est envoyé du même type que précédemment.

1.1.4.4.2. Retrait d'une Entrée - Remove Entry. [X.511]

Cette opération est utilisée pour enlever une entrée de type objet ou alias du DIT. On remarquera que les alias ne sont jamais déréférenciés et que l'entrée doit être une feuille du DIT. Le paramètre de cette fonction est le nom distingué de l'entrée à supprimer.

Si l'opération réussit, l'utilisateur en est averti. Dans le cas contraire, un message d'erreur est envoyé du même type que précédemment.

1.1.4.4.3. Modification d'une Entrée - Modify Entry. [X.511]

L'opération de modification d'entrée permet de faire une série de modifications dont voici la composition :

- ajouter un nouvel attribut.
- retirer un attribut.
- ajouter des valeurs d'attribut.
- retirer des valeurs d'attribut.
- remplacer des valeurs d'attribut.
- modifier un alias.

Les arguments de cette opération sont les suivants :

- le nom distingué de l'entrée devant subir les modifications. Un alias ne sera pas déréférencié.
- l'ensemble des modifications à effectuer. Ces modifications ne doivent pas violer le schéma du Directory. Elles sont de quatre types :
- addAttribute : Ceci identifie un nouvel attribut.
- removeAttribute : Cet argument identifie l'attribut à retirer.
- addvalues : Ce type identifie un attribut par son type et spécifie la ou les valeurs à ajouter.
- removeValues : Ceci identifie un type d'attribut et indique les valeurs d'attribut à supprimer.

Si l'opération réussit, l'utilisateur en est averti. Dans le cas contraire, une erreur de type habituel est signalée. Il faut remarquer que l'opération de modification d'entrée est atomique, càd que l'entrée garde son état initial si un des changements provoque une erreur.

1.1.4.4.4. Modification d'un Nom Distingué Relatif - Modify RDN

.

Une opération de modification de RDN est utilisée pour changer le nom relatif distingué d'une entrée. Celle-ci doit être une feuille du DIT.

Cette opération possède les paramètres suivants :

- le nom distingué de l'entrée dont le nom distingué relatif doit être modifié. On notera que les alias ne sont pas déréférenciés.
- le nouveau nom distingué relatif de cette entrée. Si une valeur d'attribut n'est pas présente dans l'entrée, elle est ajoutée si ceci est possible. Dans le cas contraire, une erreur est retournée.
- un indicateur précisant si les valeurs d'attribut de l'ancien RDN qui ne se retrouvent pas dans le nouveau RDN doivent être supprimées.

L'utilisateur est averti de la réussite ou de l'échec de l'opération.

1.1.4.5. Complément sur les Erreurs et l'Abandon. [X.511]

Le Directory ne continue pas une opération lorsqu'une erreur est détectée. Dans le cas où plusieurs erreurs seraient détectées simultanément, le Directory précise les types d'erreur rencontrés. Ces types se trouvent dans la liste ci-dessous classés par ordre décroissant d'importance :

- NameError.
- UpdateError.
- AttributeError.
- SecurityError.
- ServiceError.

On signalera que seulement l'erreur la plus importante est signalée. Les erreurs suivantes ne respectent pas ce schéma :

- AbandonFailed parce qu'elle est spécifique à une opération d'abandon qui ne peut avoir d'autre erreur.
- Abandonned qui n'est pas reporté si une autre erreur est détectée. Dans ce cas, une erreur AbandonFailed est reportée avec l'indication de l'autre erreur et comprend la mention trop tard (TooLate).
- Referral qui n'est pas une erreur mais qui indique a l'utilisateur qu'il doit présenter sa requête à un autre point d'accès.

Nous allons maintenant passer au détail de chacune des erreurs précédemment citées.

1.1.4.5.1. Abandonned. [X.511]

Une indication d'abandon peut être reportée pour toutes opérations de lecture et de recherche quand un DUA demande l'abandon d'une opération clairement identifiée.

1.1.4.5.2. Abandon Failed. [X.511]

Une indication d'échec d'abandon signale qu'un problème a été rencontré lors de l'opération d'abandon. Cette indication contient deux paramètres :

- le problème rencontré qui peut être de trois types :

- noSuchOperation qui signifie que l'on voulait annuler une opération qui n'existait pas.
- tooLate qui indique que l'opération a déjà été effectuée.

- cannotAbandon qui signifie que l'opération ne peut être interrompue (par exemple une modification).

- l'identification de l'opération abandonnée.

1.1.4.5.3. Name Error. [X.511]

Une erreur de nom rapporte un problème relatif au nom passé comme argument à une opération. Les paramètres de cette opération ont la signification suivante :

- le problème rencontré qui peut être de quatre types différents

- noSuchObject signifiant que le nom n'identifiait aucune entrée.
- aliasProblem indiquant qu'un alias a été déréférencié et ne pointait sur aucune entrée.
- invalidAttributeSyntax signifiant qu'un type d'attribut et sa valeur étaient incompatibles avec le nom de l'entrée.
- aliasDereferencingProblem stipulant qu'un alias a été rencontré dans une situation interdisant les alias.

- un paramètre contenant le nom de l'entrée la plus basse du DIT collant avec le nom passé en paramètre.

1.1.4.5.4. Update Error. [X.511]

Une erreur de mise-à-jour rapporte un problème relatif à un essai d'ajout, de modification ou de suppression d'information dans le DIB. Cette erreur a comme paramètre le problème encouru qui peut être du type suivant :

- namingViolation rapportant un essai de modification ou d'addition violant les règles de structure du DIT définies dans le Directory schema.
- objectClassViolation signifiant qu'un essai de mise-à-jour d'une entrée viole les caractéristiques de la classe d'objets dont dépend cette entrée.
- notAllowedOnNonLeaf indiquant un essai d'opération sur une entrée qui n'est pas une feuille du DIT.
- notAllowedOnRDN rapportant une opération affectant le RDN d'une entrée.
- entryAlreadyExists signifiant qu'essai d'ajout d'entrée déjà existante a été découvert.
- objectClassModificationProhibited signalant une opération tentant de modifier l'attribut de classe d'objet d'une entrée.

1.1.4.5.5. Attribute Error. [X.511]

Une erreur d'attribut signale un problème relatif à un attribut. Cette erreur a deux paramètres :

- le nom distingué de l'entrée impliquée.
- le type de problème rencontré accompagné du type de l'attribut et de sa valeur :

- noSuchAttributeOrValue signifie que l'entrée ne comporte pas un tel type d'attribut ou une telle valeur d'attribut.
- invalidAttributeSyntax indique qu'une valeur d'attribut ne correspond pas avec le type d'attribut.
- undefinedAttributeType résulte de l'emploi d'un type d'attribut non défini.
- constraintViolation indique qu'un attribut ou une valeur d'attribut viole une contrainte.
- AttributeOrValueAlreadyExists rapporte un essai d'ajout d'attribut ou de valeur d'attribut existant.

1.1.4.5.6. Security Error. [X.511]

Un erreur de sécurité rapporte l'échec d'une opération pour des raisons de sécurité. Elle a un seul paramètre, le problème rencontré, qui peut être :

- inappropriateAuthentication signifie que le niveau d'authentification donné par le demandeur n'est pas suffisant.
- invalidCredentials survient quand l'identité de l'utilisateur est fausse.
- insufficientAccessRights indique à l'utilisateur qu'il n'a pas assez de privilèges pour exécuterl'opération demandée.
- protectionRequired signifie que le Directory ne donnera les informations que si elles sont codées (signées).
- noInformation quand aucune information n'est disponible.

1.1.4.5.7. Service Error. [X.511]

Les erreurs de services rapportent les problèmes concernant l'accomplissement des requêtes. Ce message d'erreur contient chaque fois la raison de celle-ci qui peut être entre autres :

- busy indique que le Directory est trop occupé pour satisfaire la requête.
- unavailable signale que des parties du Directory ne sont pas disponibles pour le moment.
- unwillingToPerform indique que le Directory ne veut pas satisfaire la requête car elle va consommer trop de ressources.
- unableToProceed signifie que le DSA n'est pas capable d'associer une entree au nom reçu (Naming Resolution) parce que celui-ci ne correspond pas à son contexte de denomination (voir 1.1.5.3).
- timeLimitExceeded signale que la limite de temps allouée à la requête est dépassée.
- ditError indique que le Directory ne peut accomplir la requête à cause d'un problème dans la cohérence du DIT.

1.1.5. Distribution du Directory - Distributed Directory. [X.518]

1.1.5.0. Introduction.

Le service abstrait du Directory permet l'interrogation, la récupération et la modification des informations contenues dans le DIB. Ce service est décrit abstraitement dans la section 1.1.4.

Cette spécification abstraite ne s'attarde en aucune façon sur la réalisation physique du Directory. De plus elle n'indique pas si le Directory est centralisé (i.e. un DSA) ou distribué en un certain nombre de DSAs.

Cette section expliquera donc le raffinement de ce service abstrait dans le cas d'un Directory distribué.

1.1.5.1. Modèles de Directory Distribués. [X.518]

1.1.5.1.0. Introduction. [X.511]

Comme le spécifie le service abstrait du Directory, le Directory est un objet offrant des services à ses utilisateurs via des ports, chacun de ces ports fournissant un ensemble de services. Les utilisateurs du Directory accèdent à ces services par des points d'accès. Le Directory peut avoir un ou plusieurs points d'accès et chacun d'eux est caracterisé par les services qu'il offre et le mode d'interaction utilisé pour les fournir.

Cette section spécifie la structure interne du Directory quand celui-ci est distribué. La figure 1.6 bis illustre un modèle de Directory distribué.

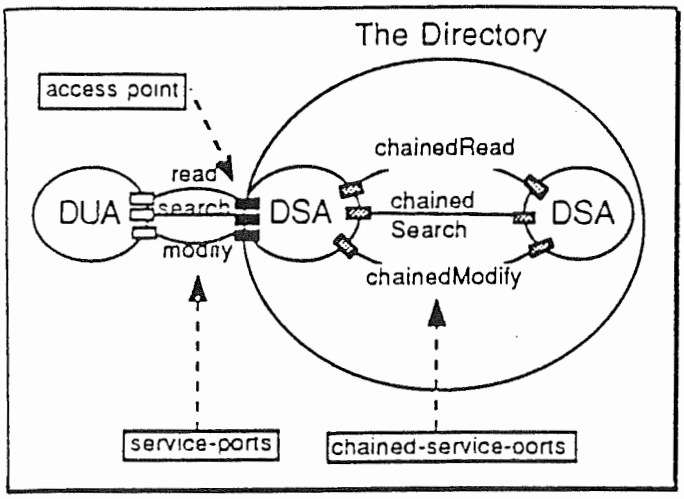


Fig 1.6bis : Les objets du Directory distribué.

Les DSAs sont définis dans le but de réaliser une distribution du DIB et d'interagir d'une manière coopérative pour fournir aux utilisateurs (DUAs) les services du Directory.

Les DSAs, comme le Directory sont caractérisés par leurs ports externes visibles. Ces ports sont de deux types :

- les ports de service (Service Ports) qui sont identiques à ceux décrits précédemment dans la section 1.4. de ce chapitre càd le readPort, le modifyPort et le searchPort. Les ports de service d'un DSA sont en fait les points d'accès au Directory.
- les ports de service chainés (Chained Service Ports) qui permettent l'intercommunication entre DSAs afin de pourvoir aux services offerts par le Directory. Ce sont le chainedReadPort, le chainedModifyPort et le chainedSearchPort.

1.1.5.1.1. Modèle d'Interaction des DSAs. [X.518]

Une des caractéristiques de base du Directory est qu'étant donné un DIB distribué, un utilisateur est capable d'obtenir une réponse à sa requête et ceci peu importe le point d'accès auquel la requête a été soumise.

Pour réaliser cela, il est nécessaire que les DSAs aient une certaine connaissance de l'endroit où est située l'information demandée (voir 1.1.5.1.3.).

Trois modes d'intéraction des DSAs ont été définis afin de remplir ses obligations. Ce sont le chainage, le multi-casting et le referral.

1.1.5.1.1.1. Chaining. [X.518]

Le chainage (illustré à la figure 1.7.) peut être utilisé par un DSA pour passer la requête à un autre DSA quand le DSA initiateur a des connaissances sur le contexte de dénomination tenu par le DSA récepteur.

Le contexte de dénomination (Naming Context) est un sous-arbre du DIT.

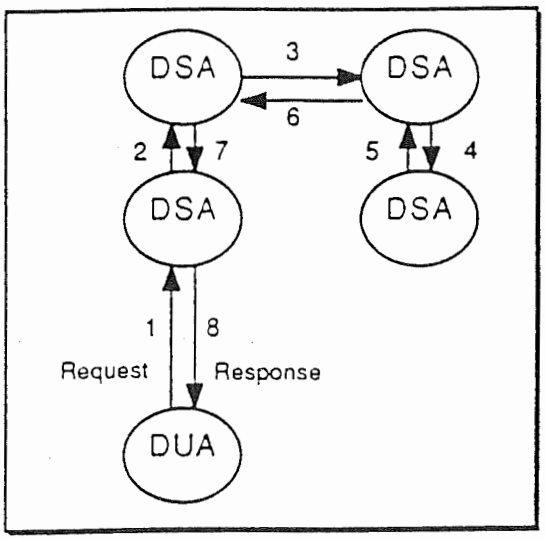


Fig 1.7. Chainage.

1.1.5.1.1.2. Multi-casting. [X.518]

Le multi-casting est un mode d'interaction pouvant être utilisé par un DSA pour chainer une requête identique en parallèle ou séquentiellement vers un ou plusieurs DSAs quand le DSA initiateur ne connait pas le contexte de dénomination complet tenu par les autres DSAs.

Chacun des DSAs recevant la requête va essayer de la satisfaire en effectuant la résolution de dénomination (Naming Resolution) mais au maximum un seul sera capable de faire progresser la requête. Tous les autres renverront l'erreur de service UnableToProceed. Ceci peut ne pas être vrai en cas de requête de recherche.

La figure 1.8. représente un exemple de multi-casting.

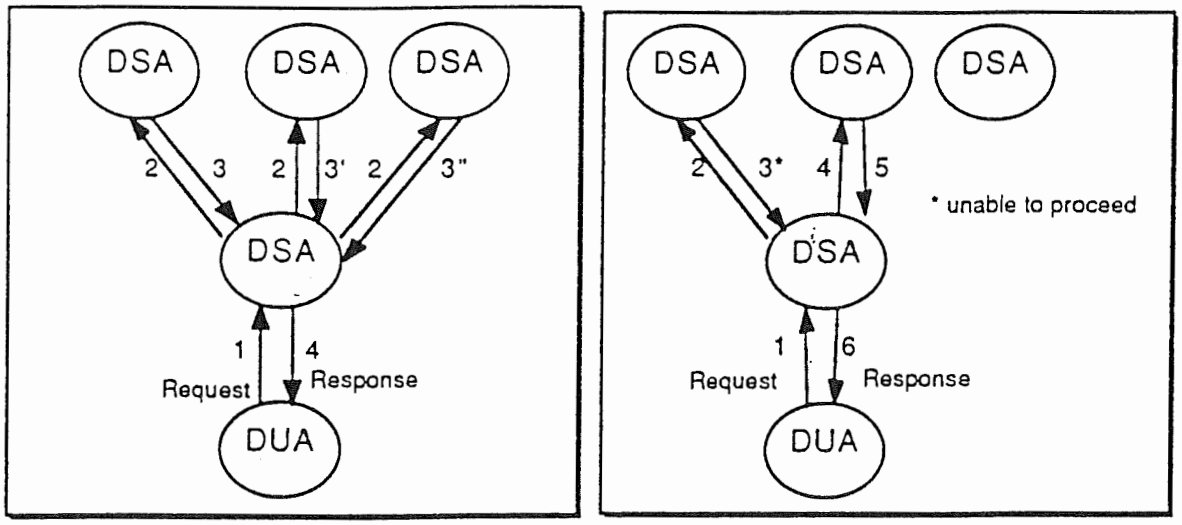


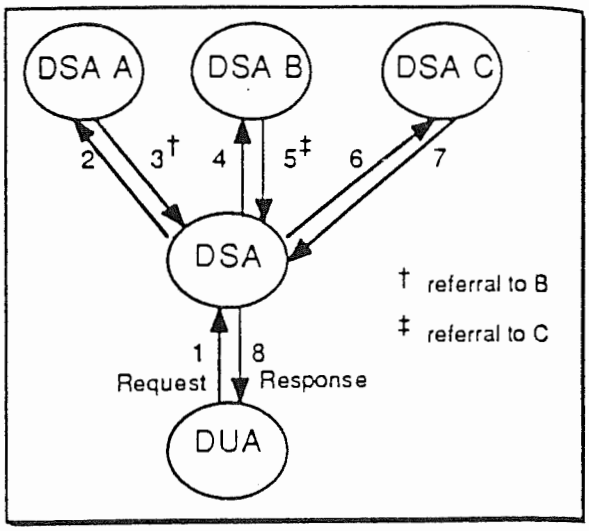
Fig 1.8. Multi-casting en parallèle. Multi-casting séquentiel.

1.1.5.1.1.3. Referral. [X.518]

Une référence (Referral), illustrée par les figures 1.9, est retournée par un DSA dans sa réponse à une requête demandée soit par un autre DSA (dans ce cas, les deux DSAs ont un port de service chainé), soit par un DUA. La référence peut être la réponse complète à la requête et est considérée comme une erreur, soit elle peut être une partie de la réponse.

Le DSA (figure 1.9a) recevant la référence peut utiliser l'information contenue dans celle-ci pour continuer l'opération par chainage ou par multi-casting.

Un DSA (figure 1.9b) recevant une référence peut l'inclure dans sa réponse . Un DUA recevant celle-ci peut utiliser l'information qui y est contenue pour contacter un ou plusieurs autres DSAs afin de progresser dans la requête.



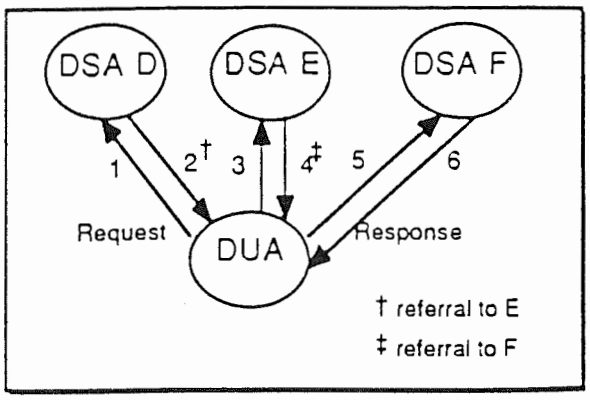


Fig 1.9a Referral (DSA) 1.9b Referral (DUA).

1.1.5.1.2. Distribution du Directory. [X.518]

Chaque DSA dans le Directory contient une partie du DIB. Les fragments du DIB tenus par un DSA sont décrits en terme de DIT et comprennent un ou plusieurs contextes de dénomination. Rappelons que le contexte de dénomination (Naming Context) est un sous-arbre du DIT.

Il est possible pour un administrateur de DSA d'avoir une autorité adminisatrative sur plusieurs contextes de dénomination disjoints. Pour chacun de ceux-ci, le DSA doit retenir la séquence de RDNs qui mène de la racine (Root) du DIT à la racine du sous-arbre comprenant le contexte de dénomination. C'est ce qu'on appelle le préfixe du contexte (Context Prefix).

La figure 1.10. représente un hypothétique DIT partitionné en cinq contextes de dénomination (A, B, C, D et E) qui sont distribués physiquement en trois DSAs (DSA1, DSA2 et DSA3).

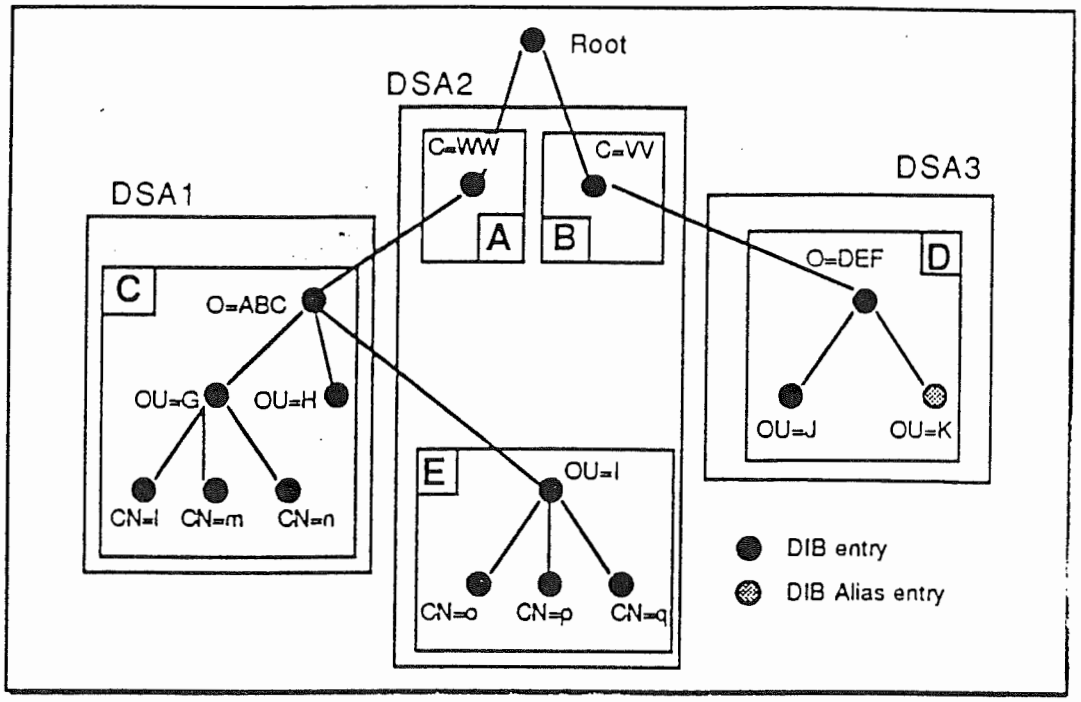


Fig 1.10. Un hypothétique DIT.

On remarquera que la racine du DIT n'est tenue par aucun DSA.

1.1.5.1.3. Connaissance (Mémoire des DSAs). [X.518]

Le DIB est distribué en plusieurs DSAs qui contiennent donc chacun une partie du DIB. Bien entendu, la distribution du Directory doit être rendue transparente. Pour l'utilisateur, chaque DSA contient le DIB tout entier.

Dans le but de satisfaire les requêtes que l'utilisateur propose aux DSAs, il est nécessaire que ceux-ci soient capables d'interagir avec les autres DSAs contenant d'autres parties du DIB. Nous allons donc parler dans cette section des connaissances nécessaires aux DSAs pour "coller" un emplacement sur un nom (nom distingué paramètre de l'opération).

Conceptuellement, les DSAs contiennent deux types d'information :

- une partie du DIB bien évidemment.
- un ensemble de connaissances (Knowledge Information) qui sont détaillées ci-dessous.

1.1.5.1.3.1. Références minimales (Minimal Knowledge References)

.

Comme on l'a dit précédemment, une des principales propriétés du Directory est d'offrir l'accès à une entrée du DIB indépendemment de l'endroit où a été générée la requête. Afin d'accomplir ceci, chaque DSA doit contenir au moins les références suivantes :

- les références subordonnées. (1.1.5.1.3.3.)
- les références supérieures. (1.1.5.1.3.4.)
- les références subordonnées non-spécifiques. (1.1.5.1.3.5.)

1.1.5.1.3.2. Contexte de la racine du DIT (Root Context). [X.518]

Nous avons deja remarqué que la racine du DIT n'était tenue par aucun DSA. Ceci dit, la fonctionnalité d'un "DSA-racine" concernant la résolution des noms (Name Resolution) doit être effectuée par les DSAs qui ont un contexte de dénomination directement subordonné à la racine. Ces DSAs portent le nom de DSAs de premier niveau (First Level DSAs). Chaque DSA de ce type doit être capable de simuler la fonctionnalité d'un DSA-racine. Pour cela, ces DSAs doivent posséder des connaissances sur le contexte de dénomination de la racine. Ce contexte est donc dupliqué dans chacun des DSAs de premier niveau.

Cette façon de procéder rejoint la propriété d'accès à n'importe quelle entrée indépendemment du point d'accès car :

- chaque DSA de premier niveau contient le contexte de dénomination de la racine, ce qui implique un chemin de référence pour tous les autres DSAs de ce type.
- chaque DSA n'étant pas de ce type contient la référence de son contexte de dénomination supérieur et par là même, possède un chemin de référence pour tous les DSAs de premier niveau.

1.1.5.1.3.3. Références Subordonnées (Subordinate References)

Une référence subordonnée consiste en :

- un nom distingué relatif (RDN) correspondant à une entrée immédiatement subordonnée du DIB.
- un point d'accès au DSA administrant cette entrée.

Toutes les entrées subordonnées tenues par un autre DSA doivent être représentées par des références subordonnées.

1.1.5.1.3.4. Références Supérieures (Superior References). [X.518]

Une référence supérieure consiste en un point d'accès à un DSA.

Chaque DSA n'étant pas de premier niveau retient précisément une référence supérieure. Celle-ci fera partie du chemin de référence menant à la racine du DIT.

Si un nouveau DSA de premier niveau est ajouté, il doit contenir le contexte de la racine et avertir tous les autres DSAs de ce type.

1.1.5.1.3.5. Références Subordonnées Non-spécifiques. [X.518]

Une référence subordonnée non-spécifique consiste en un point d'accès à un DSA qui contient un ou plusieurs contextes de dénomination immédiatement subordonné.

Ce type de référence est optionnel. Pour chacun des contextes de dénomination que contient un DSA, ce DSA peut aussi contenir un certain nombre de références subordonnées non-spécifiques qui seront évaluées après les autres références.

1.1.5.2. Service Abstrait des DSAs. [X.518]

Le service abstrait du Directory a été totalement défini dans la section 1.4 de ce chapitre. Cependant, lorsque le Directory est distribué, il l'est selon ses DSAs qui rendent donc des services particuliers dans ce cas. Ce service abstrait raffiné est le sujet de cette section.

1.1.5.2.1. Opérations de Connexion et de Déconnexion entre DSAs - Abstract Bind and Unbind Operations. [X.518]

Les opérations de connexion et de déconnexion aux DSAs sont utilisées par un DSA au début et la fin d'une période d'accès à un autre DSA.

1.1.5.2.1.1. DSA Bind Operation. [X.518]

Une opération de connexion à un DSA est utilisée par un DSA pour connecter son chainedRead ou son chainedSearch ou son chainedModify port à un port de même type d'un autre DSA.

Les arguments passés lors de la connexion sont les suivants :

- l'identité du DSA initiateur (Credentials) qui permet l'identification de l'entité-application (AE) émettrice. Cette identité est passée sous la forme d'un nom distingué.

Si la connexion réussit, le DSA initiateur reçoit l'identité du DSA récepteur sous la forme d'un nom distingué.

1.1.5.2.1.2. DSA Unbind Operation. [X.518]

Une opération de déconnexion à un DSA est employée par un DSA pour déconnecter un de ses ports chainés au port chainé de même type d'un autre DSA auquel il est connecté.

Cette opération n'a ni argument ni erreur.

1.1.5.2.2. Opérations Abstraites Chainées - Chained Abstract Operations. [X.518]

Comme déjà dit précédemment, à chaque port du service abstrait du Directory correspond un port de DSA qui permet au service abstrait d'être réalisé en collaboration avec les autres DSAs. En ce qui concerne les opérations de ces différents ports, elles suivent aussi la même correspondance. Ces opérations sont :

- pour le chainedReadPort :
    - chainedRead.
    - chainedCompare.
    - chainedAbandon.
- pour le chainedSearchPort :
    - chainedList.
    - chainedSearch.

- pour le chainedModifyPort :
  - chainedAddEntry.
  - chainedRemoveEntry.
  - chainedModifyEntry.
  - chainedModifyRDN.

Les arguments, les résultats et les erreurs des opérations abstraites chainées sont formées systématiquement de la même façon que les arguments, les résultats et erreurs de leur opération abstraite correspondante du service abstrait du Directory. Il existe cependant une exception à cette règle. Il s'agit de l'opération d'abandon chainée (ChainedAbandon) qui est décrite dans la suite de cette section.

Une opération chainée est utilisée pour propager à certains DSAs une requête issue d'un DUA effectuant une opération abstraite (non-chainée).

Chaque opération abstraite chainée contient un paramètre en plus des paramètres des opérations non-chainée. Ces paramètres sont décrits ci-dessous.

1.1.5.2.2.1. Chaining Arguments. [X.518]

Les chaining arguments sont présents dans chaque opération chainée pour convoyer à un DSA, l'information nécessaire au succès de sa part de la requête.

Cet argument est composé des éléments suivants :

- le nom du DSA à la base de la requête.
- le nom de l'entrée objet vers laquelle on se dirige.
- l'état de l'opération en cours qui comprend :

- la phase de résolution du nom qui indique à quel point cette résolution est arrivée.

Les phases sont :
- notStarted qui indique qu'aucun DSA ayant un contexte de dénomination suffisant n'a déjà été atteint.
- proceeding signifiant que cette résolution est en cours.
- completed indiquant que la résolution est terminée.

- un entier indiquant le nombre de RDNs du nom de l'entrée objet qui ont déjà été reconnus.

- la trace de l'opération qui sert à prévenir des processus bouclants et qui comprend les éléments suivants pour chaque DSA rencontré :

- le nom du DSA.
- le nom de l'entrée objet recherchée.
- l'état de l'opération.
- un booleen indiquant si les alias doivent être déréférenciés.
- si le booleen précédent a la valeur True, un entier indiquant le nombre de RDNs du nom de l'entrée vers laquelle on se dirige qui ont été générés à partir d'attributs d'entrée de type alias.
- le type de référence utilisée pour effectuer le routage jusqu'au DSA auquel on se trouve. Ce composant est une énumération de différents types de référence :
- supérieur (Superior Reference).
- subordonnée (Subordinate Reference).
- référence croisée.
- subordonnée non-spécifique (Non-specific Subordinate Ref.)
- une limite de temps endeans laquelle l'opération doit être effectuée.
- les paramètres de sécurité (voir section 1.7 de ce chapitre).

1.1.5.2.2.2. Chaining Results. [X.518]

Les Chaining Results sont présents dans le résultat de chacune des opérations chainées et permettent le retour au DSA générateur de l'opération.

1.1.5.2.2.3. Chaining Errors. [X.518]

Si une erreur est detectée lors de la requête, une des erreurs expliquées précédemment sera retournée. Celle-ci fait partie des erreurs possibles lors de l'opération de même type mais non-chainée, exceptée la DSA-Referral qui est décrite au point suivant.

1.1.5.2.2.4. Remarque sur l'Opération d'Abandon Chainé. [X.518]

Une opération d'abandon chainé est utilisée par un DSA pour prévenir un autre DSA qu'il n'est plus intéressé par le résultat de l'opération précédemment invoquée.

On remarquera qu'un DSA n'est jamais obligé d'émettre un ordre d'abandon et encore moins de le suivre. Si l'opération d'abandon réussit, un résultat est retourné et l'opération interrompue retourne une erreur du type abandonned. Dans le cas contraire, le DSA est prévenu par une erreur de type abandonFailed.

1.1.5.2.3. Erreurs Résultant d'Opérations Chainées. [X.518]

Toutes les erreurs retournées lors d'opérations chainées sont les mêmes que celles de leurs homologues non-chainées à la seule exception de la DSA-referral error qui remplace dans ce cas la referral error.

Une DSA referral error est générée par un DSA quand celui-ci ne désire plus continuer une opération par chainage ou par multi-casting. Cette erreur contient un paramètre donnant une information à l'initiateur de la requête pour reposer celle-ci a un autre DSA.

1.1.5.3. Procédures des Opérations Distribuées. [X.518]

1.1.5.3.0. Introduction.

Cette section introduit les procédures des opérations distribuées du Directory qui sont effectuées par les DSAs. Chacun des DSAs est équipé de procédures capables de remplir complètement toutes les opérations du Directory.

Dans le cas où un DSA contient l'ensemble du DIB, toutes les opérations sont effectuées par ce DSA. Dans le cas où le DIB est distribué en plusieurs DSAs, l'exécution d'une opération est fragmentée, et une partie de celle-ci peut être accomplie par chacun des DSAs.

Dans le cas d'un Directory non distribué, les opérations sont identiques.

1.1.5.3.1. Phases d'une Opération. [X.518]

Chaque opération du Directory peut être décomposée en trois phases distinctes :

- la phase de résolution des noms (Name Resolution Phase) qui est utilisée pour localiser le DSA contenant une certaine entrée.
- la phase d'évaluation (Evaluation Phase) pendant laquelle l'opération spécifiée est exécutée.
- la phase de réunion des résultats (Results Merging Phase) pendant laquelle l'ensemble des résultats est retourné au DSA demandeur.

Dans le cas des opérations Read, Compare, List, Search et ModifyEntry, la résolution des noms s'effectue sur le nom de l'objet passé en argument dans l'opération. Par contre pour les opérations AddEntry, RemoveEntry et ModifyRDN, la résolution des noms s'effectue sur le nom de l'entrée immédiatement supérieure (qui rappelons le, est obtenu en éliminant le dernier RDN du nom de l'entrée passé en argument).

Détaillons maintenant un peu plus ces trois phases.

1.1.5.3.2. Phase de Résolution des Noms - Name Resolution Phase.

La résolution des noms est un procédé séquentiel tentant de faire correspondre chacun des RDNs d'un nom (pouvant être valide ou invalide) avec le nom des sommets du DIT, en commençant logiquement à partir de la racine et en descendant ensuite dans le DIT. Etant donné le fait que le DIT est distribué en plusieurs DSAs, chacun des DSAs ne sera capable de réaliser qu'une partie de la résolution du nom.

Quand un DSA atteint la frontière de son contexte de dénomination, il est capable de dire si la résolution peut être continuée par un autre DSA ou si le nom donné était invalide, ceci grace à l'information contenue dans son contexte de dénomination.

1.1.5.3.3. Phase d'Evaluation - Evaluation Phase. [X.518]

Quand la phase de résolution des noms est terminée, l'opération demandée peut être effectuée.

Les opérations n'impliquant qu'une seule entrée peuvent être exécutées entièrement dans le DSA contenant l'entrée. Ces opérations sont : Read, Compare, AddEntry, RemoveEntry, ModifyRDN et ModifyEntry.

Par contre, les opérations impliquant de multiples entrées doivent aussi localiser les subordonnés de l'entrée qui peuvent ou ne peuvent pas se trouver dans le même DSA. Si elles ne s'y trouvent pas, l'opération doit être dirigée vers les DSAs spécifiés par ces références subordonnées pour terminer la procédure d'évaluation. Ces opérations sont List et Search.

1.1.5.3.4. Phase de Réunion des Résultats - Results Merging.

La phase de réunion des résultats commence dès que des résultats provenant de la phase d'évaluation ont été trouvés.

Dans le cas où l'opération n'affecte qu'une seule entrée, le résultat de l'opération peut être retourné au DUA tout simplement. Par contre, quand plusieurs entrées de multiples DSAs sont affectées, les résultats doivent être regroupés.

Les réponses possibles retournées à l'initiateur de l'opération sont :

- un résultat complet.
- un résultat incomplet car certaines parties du DIT n'ont pas été visitées. Un tel résultat partiel contient aussi des références utiles à la continuation de l'opération dans les parties inexplorées du DIT.
- une erreur.
- si le demandeur était un DSA, un chaining result.

1.1.5.3.5. Fonctionnement des DSAs - Cas Distribué. [X.518]

Un DSA doit agir en accord avec des procédures bien définies afin qu'une réponse appropriée soit retournée pour chaque opération invoquée par le demandeur. Cette section spécifie le fonctionnement des DSAs comme un ensemble contenant ces procédures.

Chaque DSA peut être vu comme un processus supporté par un ensemble de procédures. La figure 1.11 illustre une vue interne du fonctionnement d'un DSA.

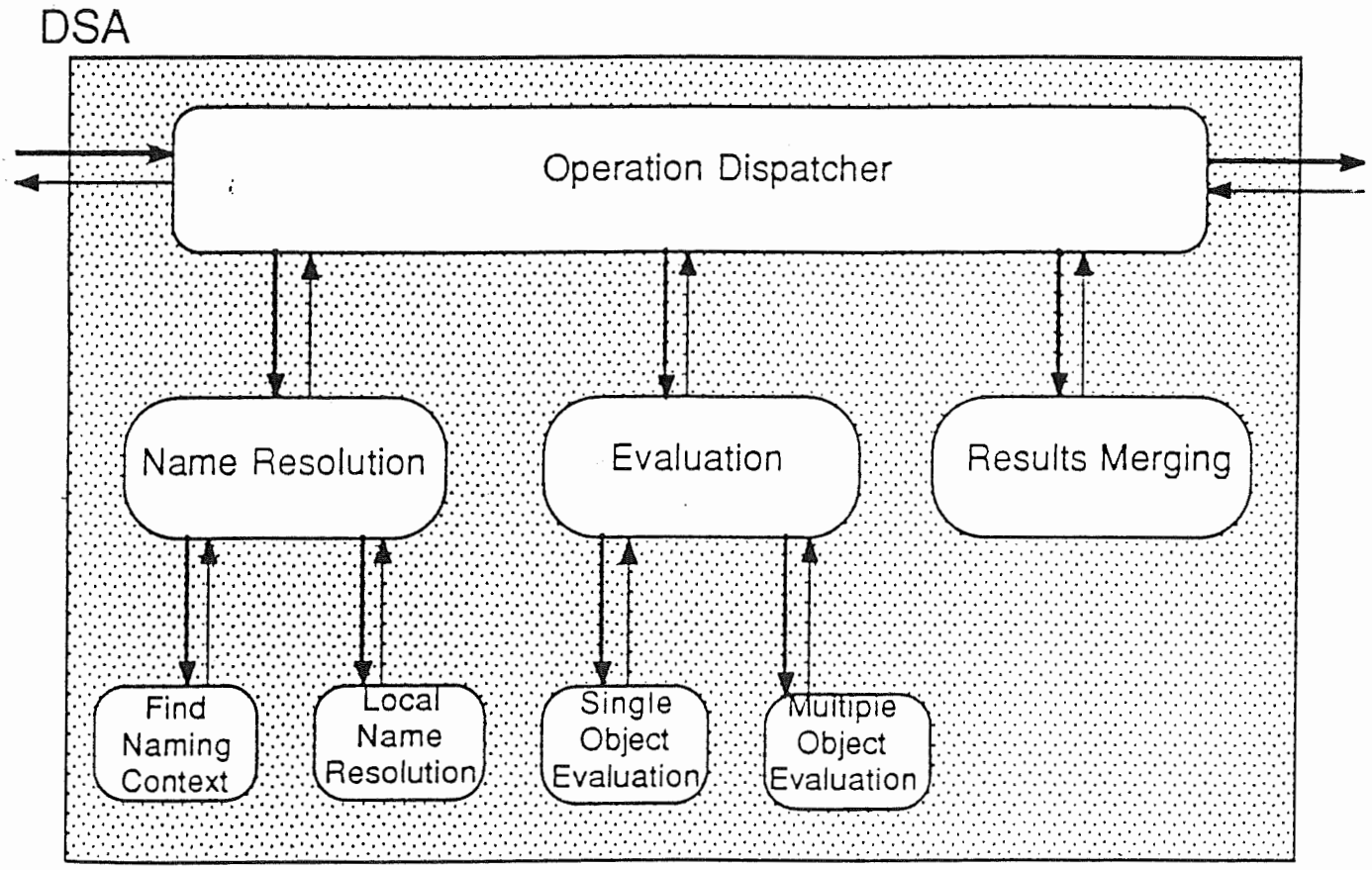


Fig 1.11. Fonctionnement d'un DSA.

L'opération dispatcher est la principale procédure de contrôle d'un DSA. Il guide chaque opération à travers les trois phases décrites au point 1.1.5.3.1.

Les procédures qui aident l'opération dispatcher sont :

- résolution des noms (Name Resolution).
- recherche du contexte de dénomination (Find Naming Context).
- résolution locale des noms (Local Name Resolution).
- évaluation (Evaluation).
- évaluation d'un objet simple (Simple Object Evaluation) .
- évaluation de multiples objets (Multiple Object Evaluation).
- réunion des résultats (Results Merging).

La figure précédente montre les relations entre ces procedures.

1.1.5.3.5.1. Opération Dispatcher. [X.518]

Recevant une opération, l'opération dispatcher la valide en vérifiant qu'elle ne boucle pas et qu'elle provient d'un utilisateur valide. Si tout se passe bien, l'opération dispatcher appelle la procédure de résolution des noms qui lui retourne, soit une erreur, une référence ou l'indication de l'appartenance de l'entrée à ce DSA.

La référence donne lieu à un chainage ou à un referral ou à un multi-casting. L'indication d'appartenance active la procédure d'évaluation.

Les résultats de provenance interne ou externe sont regroupés par la procédure de réunion des résultats.

La figure 1.12 représente un algorithme possible de l'opération dispatcher que nous donnons ici titre indicatif.

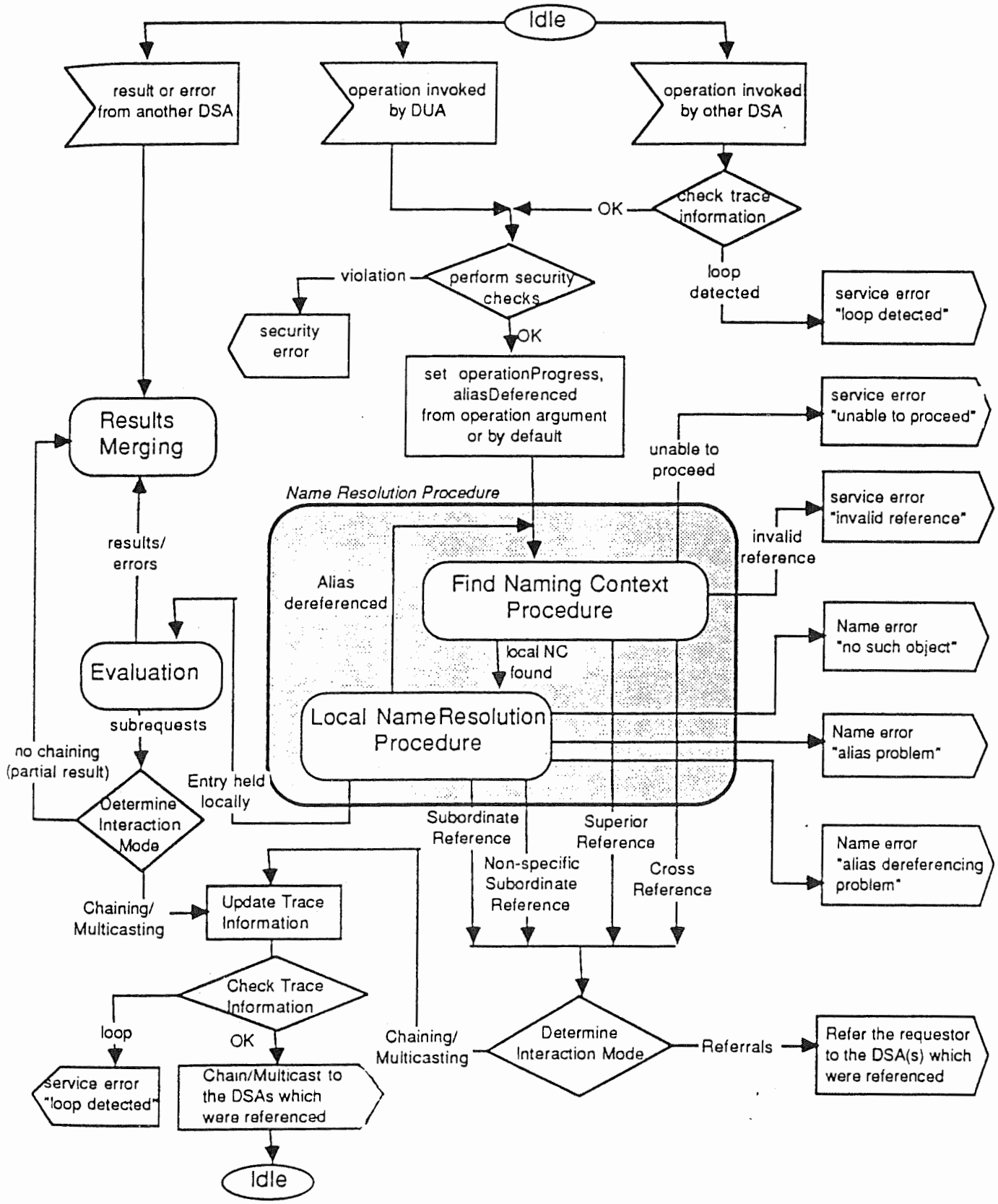


Fig 1.12. L'opération dispatcher.

1.1.5.3.5.2. La Procédure de Résolution des Noms. [X.518]

La procédure de résolution des noms appelle une procédure permettant de trouver le contexte de dénomination. Si le contexte retourné est local, la procédure de résolution locale des noms est appelée. Sinon, la procédure de résolution des noms renvoie une erreur ou une réference.

1.1.5.3.5.3. La Procédure de Recherche du Contexte de Dénomination.

La procédure permettant de trouver le contexte de dénomination essaye de faire coller le nom reçu avec les préfixes du contexte (voir 1.1.5.1.2.). Si aucun de ces préfixes ne colle, la procédure essaye d'identifier une référence croisée ou une référence supérieure. Par contre si un des préfixes colle, la procédure retourne une référence croisée correspondant à un contexte situé plus bas dans le DIT ou une indication affirmant qu'un contexte adéquat a été trouvé localement. Dans ce cas la phase de résolution des noms est entamée.

La figure 1.13 illustre un algorithme permettant de trouver le contexte de dénomination que nous citons à titre indicatif car ce seul algorithme pourrait constituer un chapitre de ce mémoire s'il était détaillé..

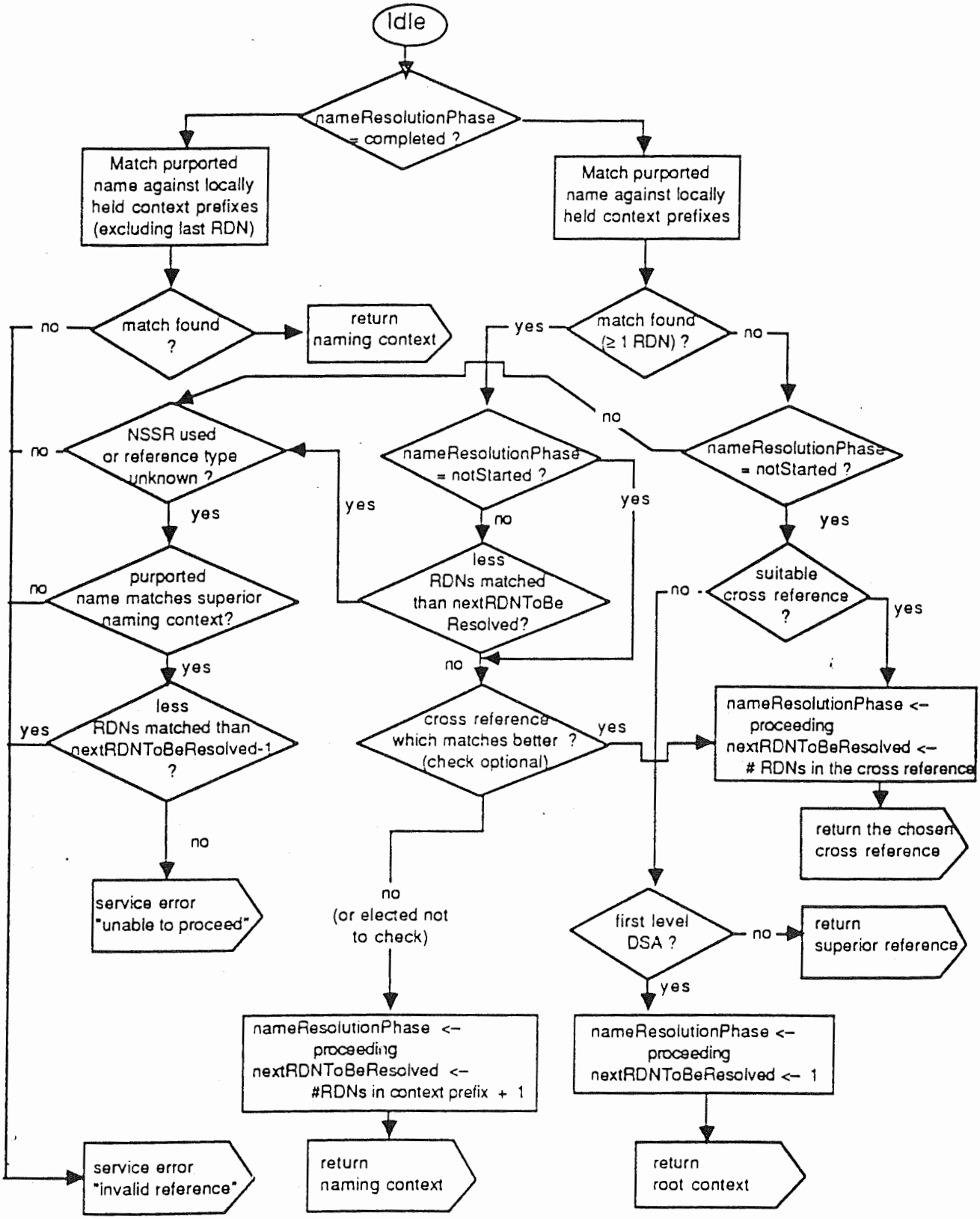


Fig 1.13. Procédure de recherche du contexte de dénomination.

1.1.5.3.5.4. La Procédure de Résolution Locale des Noms. [X.518]

La procédure de résolution locale des noms essaye de faire coller des RDNs avec le nom reçu jusqu'au moment où elle trouve l'entrée correspondante (si celle-ci se trouve dans ce DSA). Si elle ne réussit pas à faire correspondre tous les RDNs, elle essaye d'identifier des références subordonnées et puis des références subordonnées non-spécifiques et retourne celles-ci à la procédure de résolution des noms.

Si un alias est rencontré, il est déréférencé si rien ne l'interdit et cette indication est renvoyée à la procédure de résolution des noms. Par contre, si elle ne peut déréférencier l'alias, une erreur de nom est retournée.

La figure 1.14 illustre un algorithme de résolution locale des noms (donné à titre illustratif).

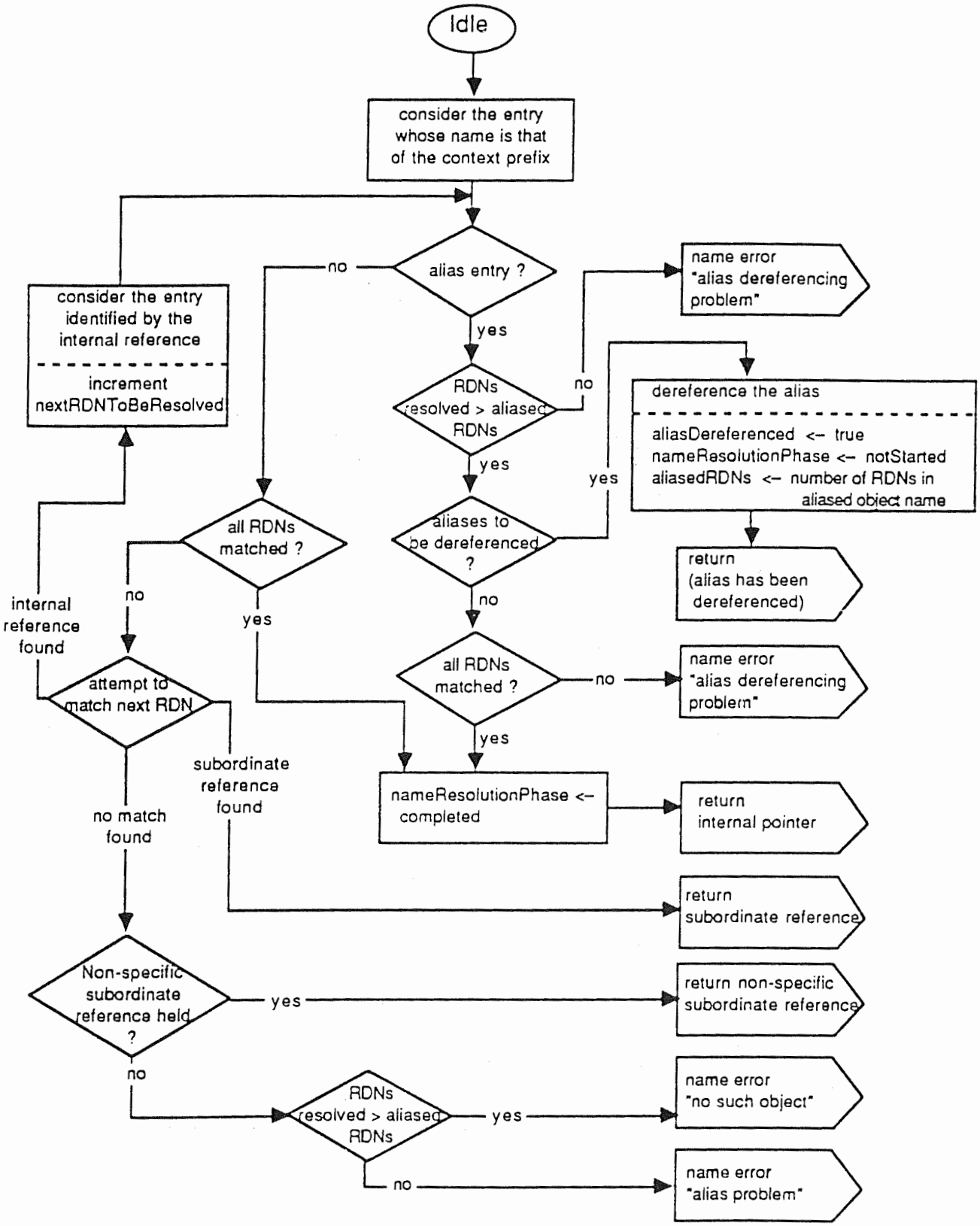


Fig 1.14. Procédure de résolution locale des noms.

1.1.5.3.5.5. Procédure d'Evaluation. [X.518]

La procédure d'évaluation exécute réellement l'opération demandée sur l'entrée trouvée. Le type d'opération permet de choisir quelle procédure est appelée pour ce travail : la procédure d'évaluation simple ou la procédure d'évaluation multiple.

1.1.5.3.5.6. Procédure de Réunion des Résultats. [X.518]

La procédure de réunion des résultats collecte les résultats et erreurs reçus des autres DSAs et les regroupe avec les résultats trouvés localement.

1.1.6. Les Protocoles - Directory Protocol. [X.519]

1.1.6.0. Introduction et Modèle de Protocoles du Directory. [X.511] [X.518] [X.519]

La recommandation X.511 définit le service abstrait entre un DUA et le Directory pour offrir des services à l'utilisateur du Directory. Le Directory est consideré comme étant un DSA qui contient le point d'accès concerné.

La recommandation X.518 définit les interactions entre deux DSAs à l'intérieur du Directory supportant les requêtes de l'utilisateur qui sont chainées. Ces deux concepts sont illustrés par la figure suivante :

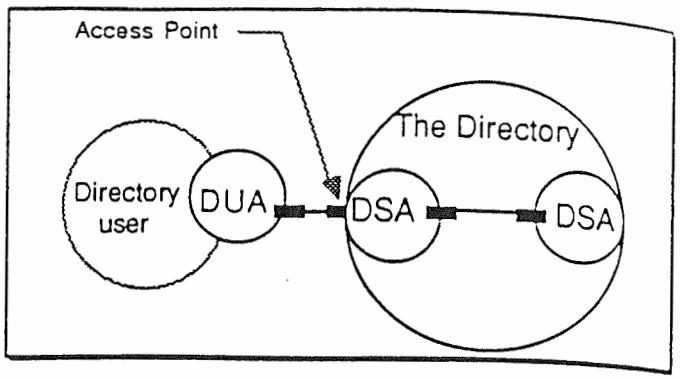


Fig 1.15. Intéractions du Directory.

Quand un DUA est situé dans un système ouvert différent de celui du DSA avec lequel il converse, ces interactions sont supportées par le protocole d'accès au Directory (DAP) qui est un protocole d'application ISO. De la même façon, quand un couple de DSA interagissants sont situés dans deux systèmes ouverts différents, les interactions sont supportées par le protocole système du Directory (DSP) qui est un protocole d'application ISO.

Le DAP et le DSP sont des protocoles qui permettent la communication entre deux processus applications. Ceci est representé dans le modèle ISO par un couple d'entités-application (AEs) utilisant les services de la couche présentation. La fonction d'une AE est de fournir un ensemble d'éléments de service-application. (ASEs). L'interaction entre deux AEs est décrite en terme de services offerts par les ASEs. Les deux ASEs communs aux protocoles du Directory sont :

- ROSE (Remote Operations Service Element) qui supporte les opérations de question - réponse qui arrivent aux ports du modèle abstrait.
- ACSE (Association Control Service Element) qui supporte l'établissement et la libération d'une association entre deux AEs. Une association entre un DUA et un DSA ne peut être établie que par le DUA et lui seul peut relacher l'association.

1.1.6.1. Directory Access Protocol - DAP.

Le DAP est utilisé pour réaliser le service abstrait du Directory (Directory Abstract Service). Le DAP comprend trois ASEs spécifiques au Directory en addition de ROSE et de ACSE qui sont:

- read ASE qui supportent les opérations abstraites du readPort a savoir read, compare et abandon.
- search ASE pour les opérations list et search.
- modify ASE pour les opérations addEntry, removeEntry, modifyRDN et modifyEntry.

1.1.6.2. Directory System Protocol - DSP. [X.518] [X.519]

Le DSP est utilisé pour réaliser les opérations distribuées décrites à la section 1.1.5.1. de ce chapitre. Le DSP comprend trois ASEs spécifiques au Directory en addition de ROSE et de ACSE qui sont :

- chained read ASE qui supporte les opérations abstraites du chained readPort à savoir chained Read, Compare et Abandon.
- chained search ASE pour les opérations chained list et search.
- chained modify ASE pour les opérations chained addEntry, removeEntry, modifyRDN et modifyEntry.

1.1.7. Sécurité et Authentification. [X.509]

Cette recommandation spécifie la forme de l'information d'authentification contenue dans le Directory et décrit la manière par laquelle cette information peut être obtenue du Directory.

Cette section décrit deux niveaux d'authentification :

- authentification simple par l'emploi d'un mot de passe comme vérification de l'identité prétendue.
- authentification forte employant un codage de l'identité de l'utilisateur.

La majeure partie des services offerts par le Directory peut être effectuée sous le premier niveau d'authentification. Par contre certains services obligent l'emploi de l'authentification forte.

Dans la suite, on supposera que les deux partenaires désirant communiquer supportent tous les deux la même technique de codage.

1.1.7.1. Authentification Simple - Simple Authentication. [X.509]

L'authentification simple est employée afin d'obtenir une autorisation locale (d'accès à un DSA par exemple) et est basée sur :

- le nom distingué de l'utilisateur.
- un mot de passe optionnel.

L'utilisation de cette technique est principalement locale, par exemple pour l'identification mutuelle de deux entités (deux DSAs, un DSA et un DUA), et peut être réalisée de différentes manières :

- par le transfert du nom distingué de l'utilisateur et d'un mot de passe optionnel sans aucun codage pour une évaluation par le récepteur.
- par la même méthode décrite ci-dessus à laquelle on ajoute un nombre aléatoire et/ou une marque de temps (Timestamp), le tout protégé par l'application d'une fonction.
- par le transfert de l'information protégée décrite ci-dessus à laquelle on ajoute un nombre aléatoire et/ou une marque de temps, le tout reprotégé par une fonction.

On entend par fonction, une application pour laquelle f(x)= y est facile à trouver mais qui est beaucoup plus ardue à résoudre pour son inverse (f'(y) = x).

La figure 1.16. illustre ce procédé.

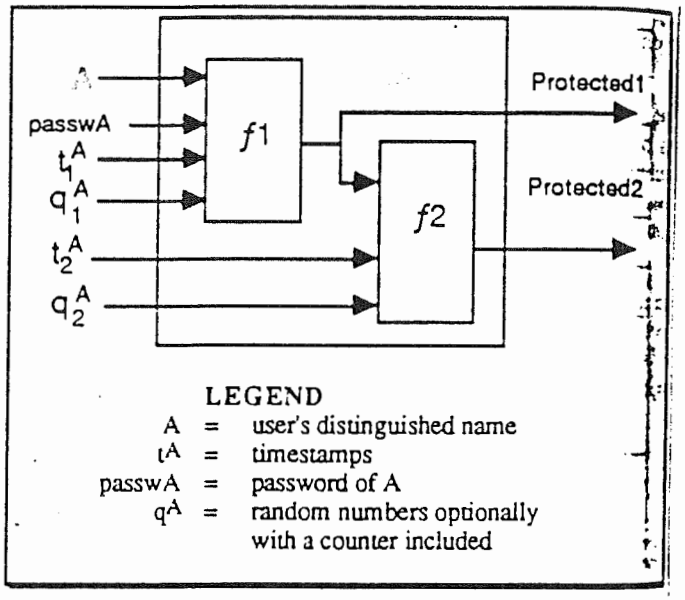


Fig 1.16. Authentification simple protégée.

1.1.7.1.1. Procédure d'Authentification Simple Non-protégée.

La procédure d'authentification simple non-protégée est représentée à la figure suivante.

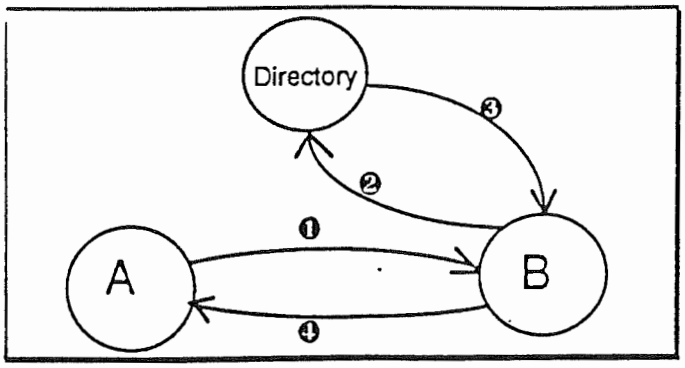


Fig 1.17. Procédure d'authentification simple non-protégée.

Cette procédure se déroule en quatre phases :

- l'utilisateur émetteur A envoie son nom distingué et son mot de passe à l'utilisateur récepteur B.
- B envoie ce qui est supposé être un nom distingué (car celui-ci peut être faux) et le mot de passe au Directory. Le mot de passe est comparé avec le celui stocké par l'attribut UserPassword au sein de l'entrée de A dans le Directory.
- le Directory confirme ou dément à B que l'identité est valide.
- le résultat de l'authentification est renvoyé à A.

1.7.1.1.2. Procédure d'Authentification Simple Protégée. [X.509]

Cette procédure se déroule en deux phases et est décrite par la figure suivante.

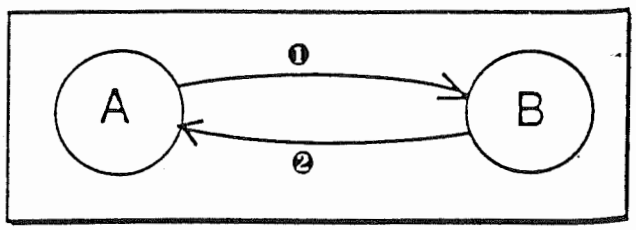


Fig 1.18. Procédure d'Authentification Simple protégée.

Dans le cas où une seule fonction est employée, les deux phases sont :

- l'utilisateur émetteur envoie son information d'identification protégée au récepteur B. La protection est le fait de l'application de la fonction au nom, au mot de passe, au nombre aléatoire et/ou à la marque de temps. L'information envoyée à B est l'information protégée plus le nombre aléatoire et/ou la marque de temps.
- B fait de même avec sa copie du mot de passe et du nom de A gardés localement et avec le nombre aléatoire et/ou la marque de temps envoyés clairement par A. Il compare les deux informations et confirme ou dément l'authentification à A.

Pour une plus grande sécurité, deux fonctions peuvent être appliquées à l'information fournie par A, mais cela ne change rien à la façon de vérifier de B. Le fait d'employer une ou deux fonctions est un accord préalable entre les deux parties.

1.1.7.2. Authentification Forte - Strong Authentication. [X.509]

L'authentification forte utilise les propriétés de la famille des systèmes cryptographiques connues sous le nom de clef publique et clef secrète. Ces deux clefs doivent permettre le codage. La clef secrète est employée pour déterminer si la clef publique a été utilisée et inversément.

Le principe d'authentification repose sur le fait que chaque utilisateur possède un nom distingué unique. Rappelons que l'attribution de ces noms est la responsabilité d'autorités de dénomination (Naming authorities).

Chaque utilisateur est identifié par la possession de sa clef privée. Un autre utilisateur est capable de déterminer si son partenaire est en possession de sa clef secrète. Pour déterminer cette possession, l'autre utilisateur doit posséder la clef publique de cet utilisateur. Il peut l'obtenir en allant lire l'information contenue dans l'entrée de cet utilisateur au sein du Directory. Cette obtention est le sujet de la section suivante.

1.1.7.2.1. Obtention de la Clef Publique d'un Utilisateur.[X.509]

Afin d'assurer une parfaite identification, un utilisateur doit obtenir la clef publique d'un autre utilisateur d'une source en laquelle il a une totale confiance. Une telle source est appelée autorité d'authentification (Certification Authority) et utilise un algorithme pour certifier une clef publique en produisant un certificat qui a les propriétés suivantes :

- tout utilisateur ayant accès à la clef publique de l'autorité de dénomination peut retrouver la clef publique certifiée d'un utilisateur.
- personne ne peut modifier le certificat sans que ceci ne soit détecté, à part l'autorité de dénomination.

L'entrée du Directory de chaque utilisateur participant à une procédure d'authentification forte contient son certificat. Ce certificat est émis par une autorité d'authentification qui est aussi une entité du DIT. La clef publique d'un utilisateur peut être retrouvée par n'importe qui connaissant la clef publique de l'autorité d'authentification de cet utilisateur.

Cependant cette façon de procéder est assez restrictive car il faut connaître la clef publique de chacune des autorités d'authentification pour pouvoir identifier une large gamme d'utilisateurs. C'est pourquoi chaque entité-autorité d'authentification Y du Directory contient un certain nombre de certificats :

- les certificats de Y générés par d'autres autorités d'authentification (Forward Certificates).
- les certificats d'autres autorités d'authentification générés par Y (Reverse Certificates).

L'existence de tels certificats permet aux utilisateurs de construire un chemin d'authentification d'un point à un autre. Un tel chemin n'est rien d'autre qu'une liste de certificats.

Les certificats sont rangés dans le Directory sous forme d'attributs de trois types différents :

- userCertificate.
- CACertificate (certificat des autorités d'authentification).
- CertificatePair (Forward et Reverse certificates).

2.1. Exemple de Modélisation des Connaissances.

L'exemple suivant illustre l'information de connaissance qui devrait être maintenue par un DSA représenté dans la figure suivante. Cette figure décrit un hypothétique DIT, logiquement partionné en cinq contextes de dénomination (A, B, C, D et E) et physiquement distribué en trois DSAs (DSA1, DSA2, DSA3). Dans l'exemple, le DSA1 maintient le contexte de dénomination C, le DSA2, les contextes A, B et E, et le DSA3 le contexte de dénomination D.

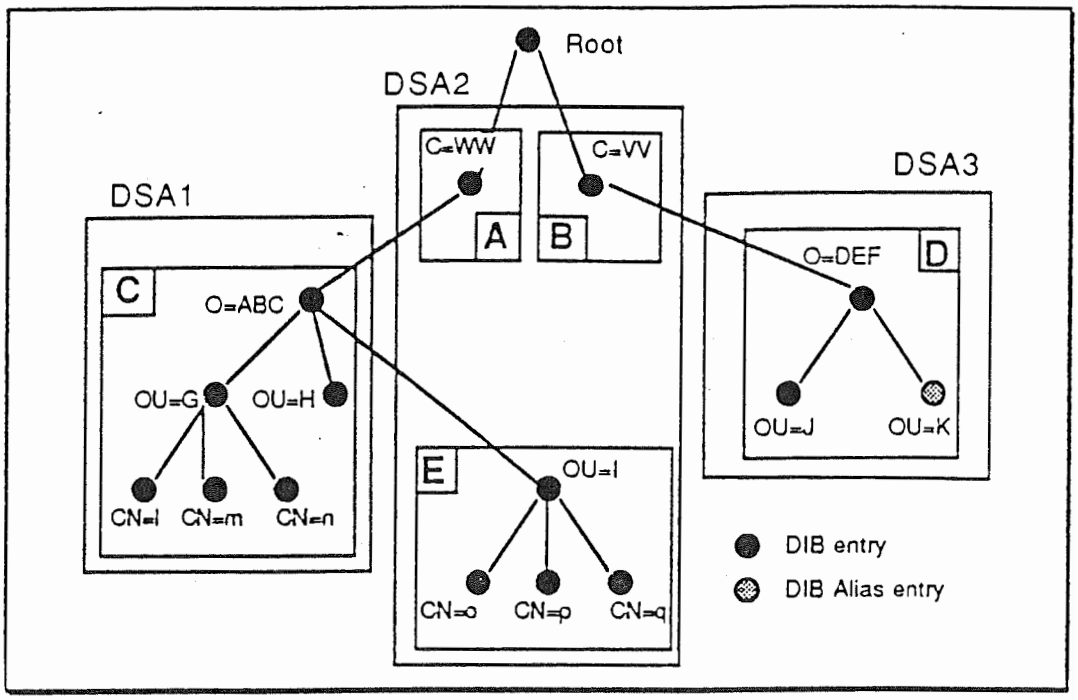


Fig 2.1 : Un DIT hypothétique.

Les abbréviations suivantes sont utilisées dans les figures suivantes :

SUPR	référence supérieure.
SUBR	référence subordonnée.
INTR	référence interne.
NSSR	référence subordonnée non-spécifique.
CROSSR	référence croisée.
DSAn	nom distingué du DSA n.
PS	adresse de présentation.
CP	préfixe de contexte de dénomination.
RDN	nom distingué relatif.
DSA	nom distingué d'un DSA.
PTR	pointeur.
AON	nom d'un objet alias.

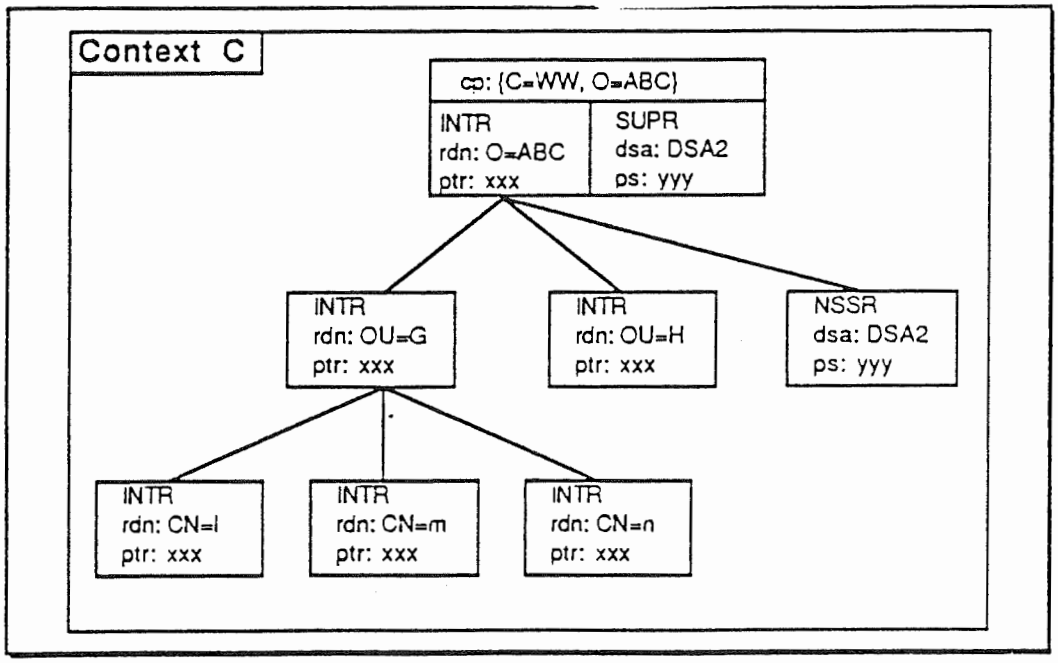


Fig 2.2: Information de connaissance du DSA1.

La figure précédente illustre l'information de connaissance qui doit être maintenue par le DSA1. Ceci inclut les préfixes de contextes de dénomination et un ensemble de références.

Préfixes de contexte de dénomination :

{C=WW, O=ABC}, contexte C

Références croisées :

{}

Références supérieures :

{DSA2, adresse de présentation du DSA2}

Références internes (pour le contexte de dénomination C) :

{C=WW, O=ABC},
{OU=G},
{OU=H},
{OU=G, CN=l},
{OU=G, CN=m},
{OU=G, CN=n}.

Références subordonnées :

{}

Références subordonnées non-spécifiques :

{DSA2, adresse de présentation du DSA2}

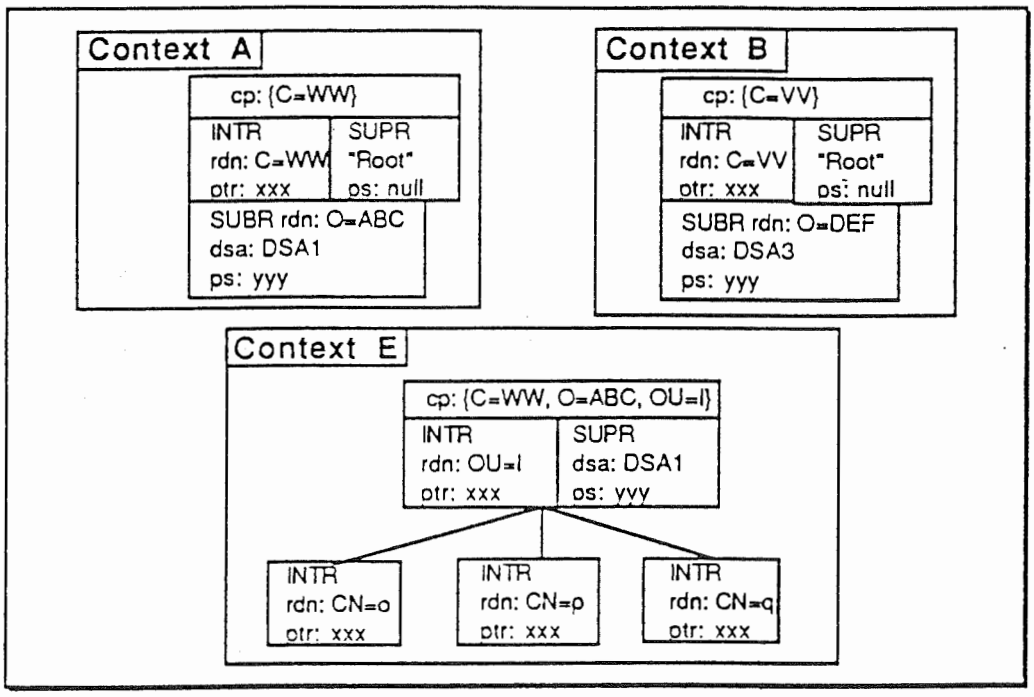


Fig 2.3 : Information de connaisance du DSA2.

La figure précédente illustre l'information de connaissance qui doit être maintenue par le DSA2. Ceci doit inclure les préfixes de contexte de dénomination et l'ensemble de références suivants :

Préfixes de contexte de dénomination :

{C=WW}, contexte A
{C=VV}, contexte B
{C=WW, O=ABC, OU=I}, contexte E.

Références croisées :

{}

Références supérieures :

{}

Références internes (pour le contexte de dénomination A) :

{C=WW}

Références internes (pour le contexte de dénomination B) :

{C=VV}

Références internes (pour le contexte de dénomination E) :

{C=WW, O=ABC, OU=I},
{CN=o},
{CN=p},
{CN=q}

Références subordonnées pour le contexte de dénomination A :

{C=WW, O=ABC}

Références subordonnées pour le contexte de dénomination B :

{C=VV, O=DEF}

Références subordonnées non-spécifiques :

{}

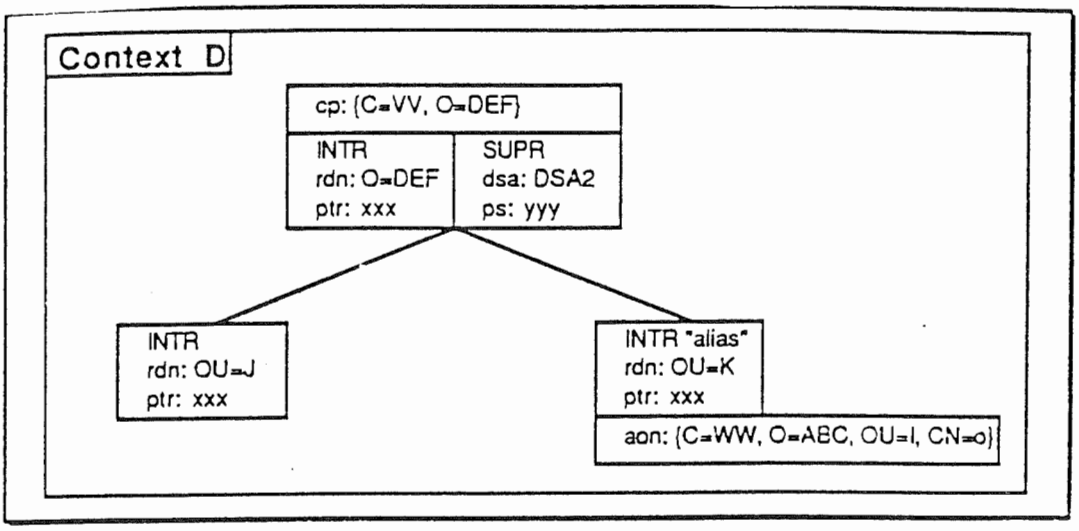


Fig 2.4 : Information de connaissance du DSA3.

Cette figure illustre l'information de connaissance qui doit être maintenue par le DSA3. Ceci doit inclure les préfixes de contexte de dénomination et l'ensemble de références suivants :

Préfixes de contexte de dénomination :

{C=VV, O=DEF}, contexte D

Références croisées :

{{C=WW, O=ABC, OU=H},
DSA1, adresse de présentation du DSA1}

Références supérieures :

{DSA2, adresse de présentation du DSA2}

Références internes (pour le contexte de dénomination C) :

{C=VV, O=DEF},
{OU=J},
{OU=K} alias de {C+WW, O=ABC, OU=I, CN=o}.

Références subordonnées :

{}

Références subordonnées non-spécifiques :

{}

2.2. Exemple de Résolution de Nom Distribué.

Ce qui va suivre est un exemple de comment la résolution des noms distribués est utilisée pour procéder à différentes requêtes du Directory. L'exemple est basé sur l'hypothétique DIT et sur la configuration des DSAs exposés précédemment.

On supposera que les requêtes sont effectuées suivant le mode de propagation chaînage et sont adressées au DSA1.

1) une requête contenant le nom distingué {C=WW ,O=ABC ,OU=G, CN=1}

◊ va s'accorder avec le préfixe de contexte de dénomination {C=WW, O=ABC} du contexte C pour lequel le DSA1 a une autorité administrative. Dès lors, la résolution des noms commencera au DSA1 avec le contexte de dénomination C.

◊ la résolution des noms procédera en descendant dans le contexte de dénomination C, en accordant avec succès chaque RDN restant, jusqu'à ce que CN=1 est rencontré.

2) une requête contenant le nom distingué {C=WW, O=JPR}.

◊ ne s'accordera pas avec un contexte de dénomination contenu dans le DSA1. Dès lors, le DSA1 utilisera ses connaissances supérieures afin de faire avancer la requête à son DSA supérieur, le DSA2.

◊ dans le DSA2, la requête s'accordera avec le préfixe de contexte de dénomination {C=WW} et le résolution des noms commencera au DSA2 avec le contexte de dénomination A.

◊ la résolution des noms ne trouvera pas de subordonné à C=WW s'accordant avec le RDN O=JPR. Dès lors, la requête va rater et le nom sera considéré comme invalide.

2) une requête contenant le nom distingué {C=VV, O=DEF, OU=K}.

◊ ne s'accordera pas avec un préfixe de contexte de dénomination maintenu par le DSA1.

◊ le DSA1 va dès lors diriger la requête vers son DSA supérieur, le DSA2.

◊ la requête s'accordera avec le préfixe de contexte de dénomination {C=VV} du contexte B maintenu par le DSA2. La résolution des noms commencera donc au DSA2 avec le contexte de dénomination B.

◊ comme la résolution des noms essaie d'accorder O=DEF, elle trouvera une référence subordonnée indiquant que {C=VV, O=DEF} est le début d'un nouveau contexte de dénomination contenu dans le DSA3.

◊ la résolution des noms continuera au DSA3 jusqu'à ce que {C=VV, O=DEF, CN=K} est trouvé.

◊ se rendant compte que les alias doivent être déférenciés, un nouveau nom sera construit en utilisant le nom "aliasé" contenu dans l'entrée {C=VV, O=DEF, CN=K}. Le nouveau nom résultant sera {C=WW, O=ABC, OU=I, CN=o}.

◊ le DSA3 va réenvoyer la requête en utilisant le nouveau nom obtenu après déréférenciation.

3.0. Introduction - Le Directory version 1992.

Ce chapitre décrit le Directory. Dans ce qui suit on trouvera une revue détaillée des nouvelles recommandations X.500 publiées en 1992. Parfois, le lecteur attentif s'apercevra que ce qui est dit dans ce chapitre et dans les suivants, va à l'encontre de ce qui a été exposé au chapitre précédent. Si cela se produit, la référence de base est ce chapitre car il expose l'évolution de ces recommandations.

Cependant, afin de ne pas se répéter tout au long de ce mémoire, nous avons décidé d'omettre tout ce qui ne changeait pas.

3.1. Modèle de l'Information Utilisateur. [X.501]

3.1.1. Les Entrées du Directory.

Les seuls changements apportés au modèle de l'information utilisateur (DIB) sont exposés dans cette section.

3.1.1.1. Les Classes d'Objets.

Les classes d'objets sont utilisées dans le Directory pour les raisons suivantes :

- décrire et catégoriser les objets et les entrées correspondants à ces objets.
- réguler, en conjonction avec les règles de structure du Directory la position des entrées dans le DIT.
- réguler en conjonction avec les règles de contenu des entrées. les attributs contenus dans ces entrées.
- identifier les classes d'entrées qui doivent être associées à une politique particulière par une autorité administrative appropriée.

Les autorités administratives seront détaillées lors de la section 3.2. tandis que les règles de contenu et de structure le seront lors de la section 3.4..

Il existe maintenant quatre types de classes d'objets :

- abstraits qui sont utilisés pour procurer les relations de base entre les sous-classes et super-classes. Il n'existe pas d'entrée de ce type au sein du Directory.
- structurels qui sont utilisés pour créer les entrées du Directory. Un objet structurel représente un élément du monde réel et porte un nom.
- alias qui sont utilisés pour donner plusieurs noms à un même objet (cfr recommandations 1988).
- auxiliaires qui ne sont pas utilisés de manière structurelle dans le Directory. Elles servent en fait à particulariser une entrée de type structurel.

3.1.1.2. Les Types d'Attributs et les Valeurs d'Attributs.

Il existe maintenant des attributs qui sont inclus dans les entrées du Directory mais qui sont invisibles aux yeux de l'utilisateur. Ces attributs portent le nom d'attributs opérationnels et sont utilisés dans des buts administratifs et opérationnels. Ces attributs sont détaillés lors de la section 3.3.4..

En ce qui concerne les valeurs d'attribut, on rajoute la définition suivante :

Une assertion de valeur d'attribut est une proposition pouvant être vraie ou fausse ou indéfinie et qui concerne la présence d'une valeur d'attribut dans une entrée du Directory.

3.1.1.3. Hiérarchies de Types d'Attributs.

Lors de la définition d'un type d'attribut, les caractéristiques de type d'attribut générique peuvent être employées comme base de cette définition. Le nouveau type d'attribut ainsi créé est un direct sous-type d'un type d'attribut générique.

L'introduction du concept de hiérarchie au sein du modèle d'attributs du Directory permet d'accéder au DIB selon différents degrés de granularité. Ceci est rendu possible par l'utilisation de types d'attribut spécifique ou générique lors de l'accès aux valeurs de ces attributs.

3.1.1.4. Règles d'Appariement - Matching Rules.

Une des capacités importantes du Directory est celle qui permet de sélectionner un ensemble d'entrées en se basant sur des assertions concernant les valeurs des attributs contenues dans ces entrées.

Le type d'assertion le plus simple est l'assertion sur la valeur d'attribut. Les assertions plus complexes utilisent les règles d'appariement. Une telle règle permet de sélectionner des entrées grâce à une assertion concernant les valeurs de leurs attributs. Une telle assertion peut être vraie, fausse ou indéfinie.

Une règle d'appariement est définie par la spécification des éléments suivants :

- une relation entre les valeurs.
- des types spécifiques des appariements supportés par cette relation.
- la syntaxe requise pour exprimer une assertion.

3.1.1.5. Attributs Collectifs.

Un ensemble d'objets peut avoir certains attributs en commun parce qu'ils peuvent partager certaines caractéristiques communes dans le monde. Un tel ensemble d'objets est appelé groupe d'objets.

Si la relation partagée constistuant un groupe d'objets est exprimable en termes de noms distingués, ce groupe est appelé groupe d'entrées et les attributs communs à ce groupe sont des attributs collectifs.

3.2. Modèle Administratif du Directory. [X.501]

3.2.0. Introduction - Aperçu Général.

L'objectif fondamental du modèle administratif du Directory est de considérer les entrées du Directory comme un ensemble bien défini afin que celles-ci puissent être gérées comme un tout. Cette section clarifie la nature et le but des autorités responsables de l'administration et les moyens par lesquels cette autorité est exercée.

Le concept de politique, défini au paragraphe 3.2.1. fournit le mécanisme par lequel les autorités administratives exercent leur contrôle sur le Directory.

3.2.1. Politique.

Une politique est une expression utilisée par une autorité administrative qui définit des buts généraux et des procédures acceptables. Une politique est définie en termes de règles qui sont suivient par le Directory et en termes d'aspects selon lesquels un utilisateur administratif, càd un utilisateur jouant le rôle d'une autorité administrative, a une certaine liberté d'action et certaines responsabilités.

Une autorité administrative définit les politiques. Il existe deux types de politique :

- Politique de domaine du DIT - DIT Domain Policy.
- Politique de domaine de gestion des DSAs - DMD Policy.

3.2.1.1. Politiques de Domaine du DIT - DIT Domain Policies.

Les politiques de domaine du DIT agissent sur le DIT. Ces politiques sont suivies par le Directory en général, dans le sens où on ne se préocupe pas des composants du Directory (i.e. DSA) contenant une partie du DIB. Ces politiques sont exprimées sous la forme d'attributs de politique.

Une politique de domaine du DIT est composée de :

- un objet (policy object) qui est une entité concernée par cette politique.

- une procédure qui est une règle définissant comment un ensemble d'objets doit être considéré et quelles sont les actions à prendre suite à ces considérations.

- des paramètres qui sont les composants d'une procédure et qui sont initialisés par une autorité administrative. Les attributs opérationnels du Directory sont utilisés pour représenter ces paramètres.

3.2.1.2. Politiques du DMD - DMD Policies.

Rappelons tout d'abord qu'un domaine de gestion du Directory (DMD) est un ensemble d'un ou plusieurs DSAs et de zéro ou plusieurs DUAs gérés par une même administration.

Une politique du DMD est une politique qui régule l'opération d'un ou plusieurs DSAs dans le DMD. Une telle politique peut s'appliquer de manière uniforme à tous les DSAs d'un DMD, à un ensemble de DSAs ou même à un DSA particulier du DMD.

Une politique du DMD peut contrôler ou restreindre le service abstrait du Directory et des DSAs. Par exemple, limiter les services de bases du Directory aux opérations d'interrogation.

3.2.2. Autorité Administrative Spécifique.

La gestion d'un domaine du DIT inclut l'exécution de trois fonctions relatives aux différents aspects de l'administration :

- gestion du sous-schèma - subschema administration.
- gestion de la sécurité - security administration.
- gestion de groupes d'entrées - entry-collection administration.

Une autorité administrative spécifique est une autorité administrative responsable d'une de ces trois gestions. Une telle autorité porte le nom d'autorité de sécurité, de sous-schéma ou de groupe d'entrées.

Plus particulièrement, une autorité de sous-schèma a pour rôle l'établissement, la gestion et l'exécution de politiques de sous-schéma qui contrôlent la dénomination et le contenu des entrées au sein d'un domaine du DIT. La structure et l'allocation des noms est la responsabilité d'une autorité de dénomination. Le rôle de l'autorité de sous-schéma est l'impléméntation des structures de dénomination dans le sous-schéma.

Une autorité de sécurité a pour rôle l'établissement, la gestion et l'exécution de politiques de sécurité qui gouvernent le comportement du Directory.

Une autorité de groupe d'entrées a pour rôle l'établissement et la gestion de groupes d'entrées au sein d'un domaine du DIT.

3.2.3. Etendues Administratives et Points Administratifs.

3.2.3.1. Etendues Administratives Autonomes.

Chaque entrée du DIT est administrée par une et une seule autorité administrative. Une étendue administrative autonome est un sous-arbre du DIT dont les entrées sont toutes gérées par la même autorité administrative.

Un domaine du DIT peut être partionné en une ou plusieurs étendues administratives autonomes distinctes.

La figure suivante représente ce concept.

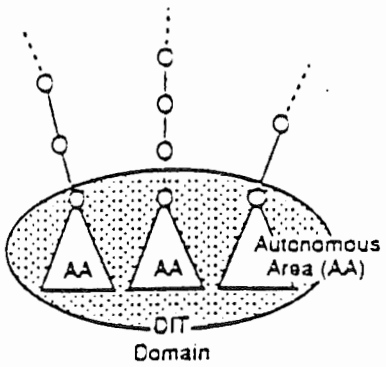


Fig 3.1. : Un domaine du DIT.

3.2.3.2. Etendues Administratives Spécifiques.

Une autorité administrative peut jouer un rôle spécifique. Si c'est le cas, une étendue administrative porte alors le nom d'étendue administrative spécifique. Il en existe trois sortes :

- étendue administrative de sous-schéma.
- étendue administrative de sécurité.
- étendue administrative de groupe d'entrées.

Une étendue administrative autonome peut être considérée comme définissant implicitement une étendue administrative spécifique pour chacun des aspects particuliers de la gestion. D'ailleurs pour chacun de ces aspects, une étendue administrative autonome peut être partionnée en plusieurs étendues administratives spécifiques distinctes.

Une autorité administrative spécifique est responsable d'une étendue administrative spécifique.

3.2.3.3. Etendues Administratives Internes.

Dans le but de la gestion de la sécurité ou des groupes d'entrées, des étendues administratives internes peuvent être définies au sein d'une étendue administrative spécifique.

Des étendues internes représentent des étendues ayant une autonomie limitée. Les entrées situées dans de telles étendues sont administrées par l'autorité admnistrative spécifique de l'étendue administrative à laquelle elles appartiennent, ainsi que par l'autorité administrative interne de l'étendue administrative interne à laquelle elles appartiennent aussi. L'autorité admnistrative spécifique a un contrôle général sur les politiques régulant ces entrées tandis que les autorités administratives internes ont un contrôle limités sur les aspects des politiques qui leurs ont été délégués.

3.2.3.4. Points Administratifs.

La spécification de la limite d'une étendue administrative autonome est implicite et consiste en l'identification d'un point dans le DIT, la racine du sous-arbre de l'étendue administrative autonome. Ce point s'appelle point administratif autonome et c'est à partir de celui-ci que débute une étendue administrative. Cette étendue continue en descendant dans l'arbre et se termine dès qu'un autre point administratif est rencontré.

Quand une étendue administrative autonome est partionnée pour un aspect spécifique de gestion, alors la spécification de la limite d'une étendue administrative spécifique consiste en l'identification d'un point, appelé point administratif spécifique à partir du quel commence l'étendue admnistrative spécifique. Celle-ci se termine dès la rencontre d'un autre de ces points.

Quand une étendue administrative autonome n'est pas partionnée, alors cette étendue correspond avec l'étendue administrative spécifique. Dès lors, un point administratif autonome est aussi un point administratif spécifique.

La spécification d'une étendue administrative interne suit le même principe (par la définition d'un point administratif interne).

La figure suivante illustre ces concepts. Une étendue administrative autonome a été partionnée en deux étendues administratives spécifiques pour un aspect particulier de la gestion. Dans une étendue administrative spécifique, une étendue administrative interne a été créée. (AP = point administratif, SAP = point administratif spécifique, IAP = point administratif interne)

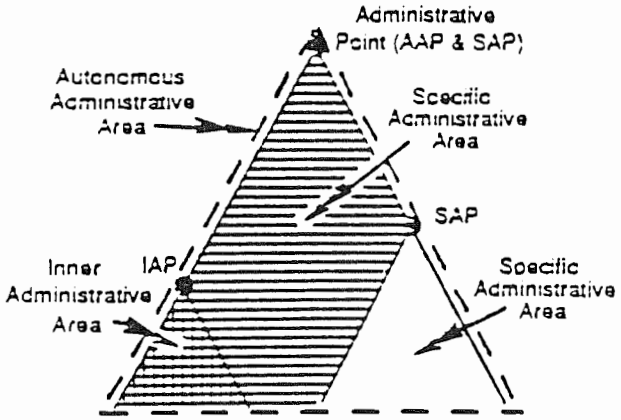


Fig 3.2. : Points administratifs et étendues.

3.2.3.5. Entrées Administratives.

A chaque point administratif du Directory est située une entrée contenant de l'information qu'utilise le Directory afin de contrôler les diverses entrées contenues dans l'étendue administrative associée à ce point. Cette entrée est dénommée entrée administrative et ces subordonnés immédiats sont en autres des sous-entrées. Ces concepts seront développés plus en détails dans la section 3.3.

Comme le décrira cette section , chaque aspect particulier d'une autorité administrative sera représenté par une sous-entrée.

3.2.4. Utilisateurs Administratifs.

Un utilisateur administratif est un représentant d'une autorité administrative. Cet utilisateur joue le rôle de l'autorité administrative. Ces rôles sont le contrôle des accès, la gestion des sous-schémas et des groupes d'entrées. L'utilisateur administratif assume son rôle en ce connectant au Directory.

3.3. Modèle de l'Information administrative et Opérationnelle.

3.3.1. Information Administrative et Opérationnelle.

L'utilisateur perçoit le DIB comme étant composé d'entrées. Ces entrées sont de deux types : entrée objet et entrée alias. Chacune de ces entrées est associée à un nom connu par le Directory. Quand on regarde le Directory selon la perspective des noms, l'information contenue dans le DIB forme le DIT.

Le modèle de l'information administrative et opérationnelle fournit un modèle général rejoignant les exigences dans les domaines suivants :

- l'information autre que les attributs utilisateurs est associée à un nom. Une telle information est exigée pour le fonctionnement du Directory lui-même. Cette information consiste en attributs collectifs et opérationnels.

- l'information associée à des unités plus grandes que les entrées : sous-arbres et sous-arbres raffinés (subtrees refinements) qui un sous-arbre qui n'est pas un arbre. L'information associée aux sous-arbres est une sous-entrée composées d'attributs.

3.3.3. Sous-Arbres (Subtrees) - Spécification - Exemple.

Lors de son fonctionnement, le Directory doit gérer divers groupes d'entrées sur lesquels agissent des politiques instaurées par des autorités administratives. Un tel groupe porte le nom de sous-arbre ou sous-arbre raffiné.

La spécification d'un sous-arbre est la définition d'un sous-ensemble d'entrées au sein d'une même étendue administrative. Cette spécification est consistuée de trois éléments qui réduisent, définissent le groupe d'entrées :

- base : la racine du sous-arbre (raffiné). Elle peut être un subordonné du point administratif ou le point administratif lui-même.
- ossature (chop) : un ensemble d'assertions concernant les noms des subordonnés de la base ou du point administratif.
- filtre : un ensemble de conditions appliquées aux subordonnés de la base ou du point administratif.

Nous n'allons pas voir ici en détails les paramètres composants de la base, de l'ossature et du filtre. Cependant, une partie de ceux-ci seront expliqués dans l'exemple suivant :

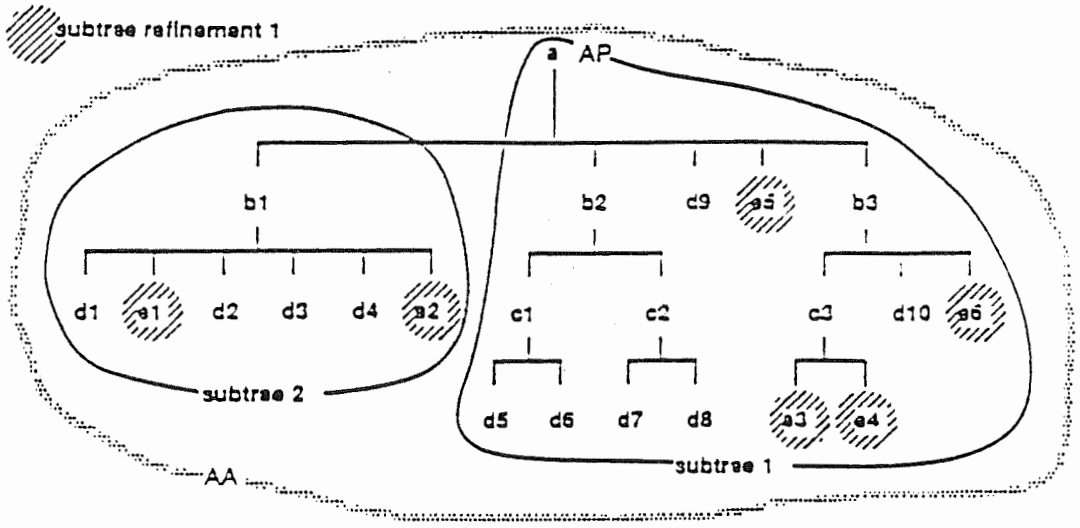


Fig 3.3. : Exemple de spécification d'un sous-arbre.

L'exemple suivant se base sur la figure ci-dessus et illustre la spécification de sous-arbres et de sous-arbres raffinés.

Soit une partie du DIT : une étendue administrative de base a. A est donc un point administratif.

Le sous-arbre 1 peut être spécifié de la manière suivante :

```
subtree1 SubtreeSpecification ::= {
   chop     exclusions { chopBefore {b1}}}
```

On remarque que la base a été omise dans la spécification. La valeur par défaut de celle-ci est donc le point administratif a. Il n'y a pas de filtre. En ce qui concerne l'ossature, il est spécifié qu'il faut exclure (exclusions) b1 et son sous-arbre (chopBefore {b1}).

Le sous-arbre 2 est spécifié de la façon suivante :

```
subtree2 SubtreeSpecification ::= {
              base    b1}
```

On sait donc que la racine de ce sous-arbre est b1. L'ossature n'étant pas définie, tout le sous-arbre de racine b1 contenu dans la même étendue admnistrative doit être considéré. Il n'y a pas de filtrage.

Supposons maintenant que les entrées e1...e6 représentent des entrées de classe organizational person. Le sous-arbre raffiné qui inclut toutes ces entrées peut être défini comme suit :

```
subtree-refinement1 SubtreeSpecification      ::=    {
   specification-filter
   item
      equality
        {objectClass, OBJECT IDENTIFIER organizational person}}
```

La base et l'ossature n'étant pas spécifiées, le sous-arbre a donc pour racine a et contient toutes les entrées contenues dans cette étendue administrative. Le filtre nous dit de prendre uniquement les entrées de classe organizational person.

3.3.4. Attributs Opérationnels et Utilisateurs.

Afin de différencier les variétés d'information associées à des noms du Directory, on peut distinguer trois niveaux d'abstraction lors de la considération des attributs :

- attributs utilisés par les utilisateurs et leurs applications qu'on nomme attributs utilisateurs.

- attributs opérationnels requis pour la régulation de l'information utilisateur tenue par le Directory ou pour tout autre but administratif.

- attributs opérationnels requis par les composants du Directory pour leur support au Directory comme application distribuée.

3.3.4.1. Attributs Utilisateurs.

Les attributs utilisateurs contiennent l'information qui est placée dans le Directory par les utilisateurs et qui est gérée par ces mêmes utilisateurs.

3.3.4.2. Attributs Opérationnels.

Les attributs opérationnels représentent une information d'un certain type concernant quelques éléments du DIB. Il existe trois types d'attributs opérationnels :

- les attributs opérationnels du Directory.
- les attributs opérationnels partagés par les DSAs (DSA-Shared)
- les attributs opérationnels spécifiques aux DSAs (DSA-Specific)

Les attributs opérationnels du Directory sont utilisés pour représenter l'information de contrôle ou toute autre information comme par exemple une indication sur le fait qu'une entrée soit ou ne soit pas une entrée "feuille".

Les deux autres types d'attributs opérationnels seront discutés ultérieurement.

3.3.5. Sous-Entrées (Subentries).

3.3.5.1. But.

Afin de contenir l'information relative aux sous-arbres (raffinés), le DIB est étendu et inclut maintenant en plus des entrées objets et alias, des sous-entrées.

Une sous-entrée est un type spécial d'entrée du DIB. Notons immédiatement que les sous-entrées ne sont pas considérées lors d'opérations de recherche ou d'énumération. Elle est liée à une entrée administrative comme étant son subordonné immédiat et elle contient des attributs qui appartiennent à un sous-arbre (raffiné) associé à un point administratif. L'information contenue dans les sous-entrées est une information opérationnelle.

3.3.5.2. Composition.

La structure d'une sous-entrée correspondant à un point administratif est illustré à la figure suivante.

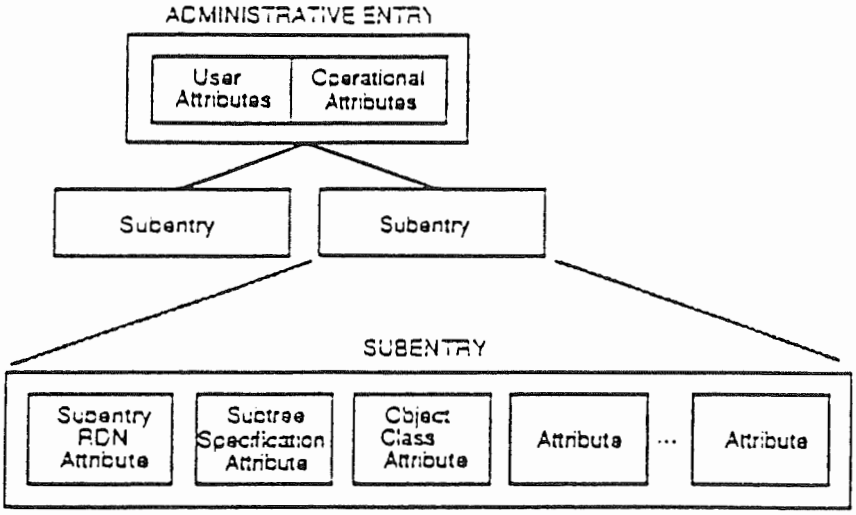


Fig 3.4. : Structure d'une sous-entrée.

Une sous-entrée est constituée de :
- un attribut de nom commum qui contient le RDN de la sous-entrée.
- un attribut de spécification du sous-arbre.
- un attribut de classe d'objet qui indique le ou les buts du sous-arbre.
- d'autres attributs qui dépendent des valeurs de l'attribut de classe d'objet.

Voyons maintenant ces attributs quelque peu plus en détail.

3.3.5.2.1. L'Attribut de Nom Commun.

L'attribut de nom commun est utilisé comme identifiant de la sous-entrée afin de distinguer celle-ci des autres sous-entrées pouvant être subordonnés immédiats d'une même entrée administrative spécifique.

3.3.5.2.2. L'Attribut de Spécification de Sous-Arbre.

Comme son nom l'indique, l'attribut de spécification de sous-arbre définit l'ensemble des entrées au sein d'une étendue administrative qui est concerné par le sous-arbre.

3.3.5.2.3. L'Attribut de Classe d'Objet.

L'attribut de classe d'objet est défini afin de réguler le reste des attributs d'une sous-entrée. Cet attribut indique quelles sont les classes suivantes qui sont associées à la sous-entrée :

- contrôle des accès de base.
- sous-schéma.
- groupe d'entrées.

3.4. Le Schéma du Directory - Directory Schema.

3.4.1. Aperçu Général.

Le schéma du Directory est un ensemble de définitions et de contraintes concernant la structure du DIT, les différentes manières d'appelation des entrées, l'information que peut contenir une entrée, les attributs utilisés pour représenter cette information, la décomposition de l'organisation en différentes hiérarchies afin de faciliter la recherche de l'information.

Formellement, le schéma du Directory comprend un ensemble de :

- définitions de name binding qui définissent les relations d'appelation élémentaires pour les objects structuraux et les alias.
- définitions des règles de structure du DIT qui définissent les noms distingués que les entrées peuvent porter et la façon dont celles-ci peuvent être en relation avec d'autres.
- définitions des règles de contenu du DIT qui étendent la spécification des attributs possibles pour les entrées d'un type structurel particulier.
- définitions des classes d'objets qui définissent l'ensemble des attributs obligatoires et facultatifs pour une entrée d'une classe particulière.
- définitions du type d'attribut qui indentifient l'objet par lequel un attribut est connu ainsi que sa syntaxe.
- définitions de la syntaxe d'un attribut en ASN.1.
- définitions de règles d'appariement.

La figure suivante résume les relations entre d'une part les définitions du schéma et d'autre part le DIT, les entrées, les attributs et les valeurs des attributs.

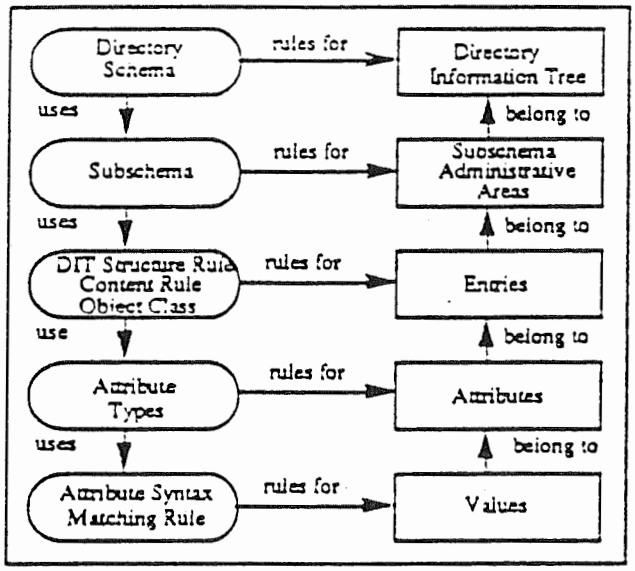


Fig 3.5. : Aperçu général du schéma du Directory.

Le schéma du Directory est distribué, tout comme l'est le DIB. Ceci se manifeste par un ensemble de sous-schémas distincts, chacun de ces sous-schémas étant régulé par une autorité administrative autonome (de type sous-schéma). C'est cette autorité qui établit les règles et les contraintes constituant le sous-schéma.

Le schéma du Directory est uniquement concerné par l'information de type utilisateur. En ce qui concerne l'information opérationnelle, un schéma spécial s'en occupe : le schéma du système du Directory. Nous ne traiterons pas de ce schéma dans ce mémoire.

3.4.2. Définition de la Structure du DIT.

3.4.2.1. Généralités.

Un aspect fondamental du schéma du Directory est la spécification de l'endroit où une entrée d'une classe particulière peut se trouver dans le DIT et comment elle doit être appelée (ce qui revient un peu au même). Il y a un certain nombre d'aspects qui doivent être considérés lors de cette spécification :

- les relations hiérarchiques des entrées dans le DIT.
- le ou les attributs utilisés pour former le RDN de l'entrée.
- le nom de l'entrée résultante.

Les deux premiers aspects seront traités au point 3.4.2.2. de ce chapitre, tandis que le troisième aspect sera détaillé au point 3.4.2.3.

3.4.2.2. Définition du Name Binding.

La définition du name binding comprend :

- l'assignation d'un objet identifiant pour le name binding.
- la spécification d'une relation de dénomination en termes de noms de classe d'objets subordonnés et supérieurs.
- l'indication des attributs devant être utilisés pour le RDN de l'entrée de cette classe d'objet.

Signalons tout de suite que le name binding ne s'applique qu'aux classes d'objets structurels et alias. Chacune de ces classes d'objets peut participer à un nombre quelconque de name bindings, mais il en faut au moins un.

Les attributs servant à la formation du RDN doivent être choisis dans une liste d'attributs devant ou pouvant se trouver dans les entrées d'une classe d'objets structurels ou alias.

3.4.2.3. Spécification des Règles de Structure du DIT.

Une règle de structure du DIT est une spécification donnée au Directory par une autorité administrative de sous-schéma. Cette règle est utilisée par le Directory pour contrôler le placement, l'agencement des entrées du DIB au sein d'un sous-schéma. Chaque entrée du DIB est régulée par une règle de structure unique du DIT. Un sous-schéma régulant un sous-arbre du DIT consistera en un ensemble de règles de structure définissant les différents types d'entrées du DIB pouvant se trouver dans ce sous-arbre.

La spécification d'une règle de structure du DIT comprend :

- un identifiant
- un indication sur le name bindng des entrées gouvernées par cette règle de structure.
- optionnellement, un ensemble de règles de structure supérieures.
- optionnellement, une indication disant si la règle est active ou non.

Un ensemble de règles de structure spécifie les formes des noms distingués acceptables au sein d'une étendue administrative de sous-schéma.

Le Directory permet à une entrée d'être en relation immédiate avec une autre si et seulement si il existe une règle de structure du DIT, contenue dans le sous-schéma, applicable.

Une règle de structure du DIT peut être active ou pas. Seulement les règles actives sont prises en considération. Le fait de garder une règle non-active est seulement une aide administrative et ceci permet de retrouver l'emplacement des entrées ayant suivi cette ancienne règle (dans un but de modification par exemple).

3.4.3. Définition des Règles de Contenu du DIT - Content Rules.

Une règle de contenu du DIT spécifie le contenu autorisé des entrées d'une classe d'objets structurels particulière par l'identification d'un ensemble de classes d'objets auxiliaires, et d'attributs obligatoires, optionnels et pre-inclus.

La spécification d'une règle de contenu comprend :

- une indication sur la classe d'objets structurels à laquelle elle s'applique.

- et optionnellement, pour les entrées régulées par cette règle

    - une indication sur les classes auxiliaires permises.
    - une indication sur les attributs obligatoires requis.
    - une indication sur les attributs optionnels permis.
    - une indication sur les attributs pré-inclus.
    - une indication sur l'état de la règle, active ou non.

Il y a précisément une règle de contenu définie par spécification de sous-schéma. De là, on peut dire que chaque entrée du Directory est administrée par une et une seule règle de contenu du DIT.

3.4.4. Définition des Classes d'Objets.

La définition d'une classe d'objet comprend :

- l'assignation d'un identifiant.
- l'indication des classes pour lesquelles cette classe peut être une sous-classe.
- l'indication du type de classe d'objet défini.
- la liste des types d'attributs obligatoires que doit contenir toute entrée de cette classe en addition de tous les types d'attributs obligatoires de ses super-classes.
- la liste des types d'attributs optionnels que peut contenir une entrée de cette classe en addition de tous les types d'attributs optionnels de ses super-classes.

Il existe un classe d'objets spéciale, dont toutes les autres classes sont sous-classes. Cette classe s'appelle Top. C'est la racine du DIT.

Il existe des restrictions dans la création de sous-classes :

- seulement une classe d'objets abstraits peut être une super-classe d'une autre de ces classes.
- seulement une classe alias peut être super-classe d'une autre classe alias.
- les classes d'objets structurels ne peuvent avoir qu'une seule autre classe d'objets structurels comme super-classe directe.
- les classes d'objets auxiliaires ne peuvent avoir que des classes d'objets auxiliaires ou abstraites comme super-classes directes.

On signalera aussi que chaque entrée contiendra un attribut indiquant de quelle classe d'objets et de quelles super-classes elle dépend.

3.4.5. Définition des Types d'Attributs.

La définition d'un type d'attribut comprend :

- l'assignation d'un identifiant.

et optionnellement :

- une indication disant si le type d'attribut est un sous-type d'attribut précédemment défini.
- une indication ou une définition de la syntaxe de l'attribut quand celui-ci n'est pas un sous-type.
- une indication sur la règle d'appariement concernant ce type d'attribut.
- une indication sur le nombre de valeurs possibles d'un attribut de ce type.
- un booléen indiquant si le type de l'attribut est collectif.
- un booléen indiquant si l'attribut est opérationnel.
- un booléen indiquant si l'attribut est modifiable par l'utilisateur.

Nous ne rentrerons pas ici dans les détails. La définition d'un type d'attribut est donnée à titre indicatif.

3.4.6. Définition des Règles d'appariement - Matching Rules.

La définition d'une règle d'appariement comprend :

- un identifiant.
- la spécification des différents types d'appariement supportés par la règle.
- la définition de la syntaxe d'une telle règle.
- la difinition des règles appropriées pour l'évaluation d'une assertion.

Encore une fois, nous ne détaillerons pas plus, étant donné que ceci n'est pas primordial à la compréhension globale du mémoire.

4. Modèle de Sécurité.

4.1. Politiques de Sécurité.

Le Directory se trouve dans un environnement où diverses autorités administratives contôlent l'accès à leur portion de DIB. De tels accès sont généralement en accord avec des politiques de sécurité contrôlées par ces autorités administratives.

La politique de sécurité comporte deux aspects différents :

- le schéma d'authentification.
- le schéma de contrôle des accès.

Nous allons parler dans la suite du schéma de contrôle des accès. Celui-ci est défini par des méthodes permettant :

- de spécifier l'information de contrôle d'accès.
- de renforcer les droits d'accès définis par cette information.
- de maintenir cette information de contrôle d'accès.

Le renforcement des droits d'accès s'applique aux accès sur :

- l'information du Directory relative à la structure du DIT.
- l'information utilisateur du Directory.
- l'information opérationnelle contenant l'information de contrôle d'accès.

4.2. Contrôle d'Accès de Base.

Cette section définit un schéma de contrôle d'accès spécifique pour le Directory. Les autorités administratives peuvent l'utiliser ou en utiliser un autre qu'elles définiraient elles mêmes.

4.2.1. Modèle de Contrôle d'Accès de Base.

Le modèle de contrôle d'accès de base du Directory définit pour chaque opération abstraite, un ou plusieurs points où prendra place le contrôle d'accès et les décisions qui s'en suivent. Chacune de ces décisions de contrôle d'accès comprend :

- le composant du Directory en cours d'accès. On l'appelle article protégé (protected item).
- l'utilisateur demandant l'opération en cours appelé demandeur (requestor).

- le droit particulier nécessaire à l'exécution d'une partie de cette opération appelé permission (permission).
- un ou plusieurs attributs opérationnels qui contiennent la politique de sécurité régulant les accès à cet article. On les appelle les articles ACI.

Dès lors, le modèle de contrôle des accès de base définit :

- les articles protégés.
- les classes d'utilisateurs.
- les catégories de permission requises afin d'exécuter chaque opération abstraite du Directory.
- l'étendue d'application et la syntaxe des articles ACI.
- l'algorithme de base, appelé fonction de décision du contôle d'accès (ACDF), qui est utilisé pour décider si un demandeur particulier a une permission d'accès en accord avec les articles ACI.

Les articles protégés regroupent les entrées, les attributs et les valeurs d'attribut. L'accès est contrôlé par des permissions accordées ou refusées. La portée des contrôles d'accès peut être une simple entrée ou un groupe d'entrées qui sont reliées au sein d'un domaine de contrôle des accès (DACD).

Les catégories de permission utilisées pour contrôler les accès à la structure du DIT sont généralement indépendantes des catégories contrôlant l'information utilisateur et opérationnelle. Depuis que chaque entrée du Directory a une position particulière au sein du DIT, l'accès à l'information utilisateur et opérationnelle implique chaque fois un accès à l'information de la structure du DIT. Dès lors, il existe deux types d'accès associés à une opération abstraite :

- accès au DIT (accès aux entrées comme éléments ayant un nom).
- accès aux attributs contenant l'information utilisateur et opérationnelle.

Pour la plupart des opérations abstraites, les permissions concernant ces deux types d'accès sont évaluées durant la procédure de décision. Ces deux formes d'accès portent le nom d'accès aux entrées et d'accès aux attributs.

Signalons immédiatement que pour exécuter des opérations sur des entrées "entières", il n'est pas nécessaire d'avoir une permission d'accès sur les attributs de ces entrées. Par contre, pour effectuer des actions sur les attributs d'une entrée, il est obligatoire de posséder les deux permissions, d'accès aux entrées et d'accès aux attributs.

4.2.1.1. Catégories de Permission d'Accès aux Entrées.

Les catégories de permission d'accès aux entrées sont :

- situer - locate.
- feuilleter - browse.
- ajouter - add.
- retirer - remove.
- modifier - modify.
- renommer - rename.

Situer, si accordé, permet aux opérations du Directory d'avoir une vue sur l'information contenue dans l'entrée. Il permet au nom distingué de l'entrée d'être révélé. Par contre, si situer est refusé, il interdit toute opération sur cette entrée.

Feuilleter, si accordé, permet aux opérations du Directory d'avoir une vue sur le nom de l'entrée. En particulier, il offre une vue sur les entrées spécifiées indirectement, en d'autres termes que le nom distingué de celles-ci.

Ajouter permet la création d'entrées au sein du Directory.

Retirer, si accordé, permet le retrait d'entrées du Directory.

Modifier permet de changer l'information contenue dans une entrée.

Renommer, si accordé, permet de renommer une entrée et tous ces subordonnés. Cependant, la permission doit être accordée pour chacun des subordonnés.

4.2.1.2. Catégories de Permission d'Accès aux Attributs.

Les catégories de permission d'accès aux attributs sont :

- comparer - compare.
- lire - read.
- feuilleter.
- ajouter.
- enlever - remove.

Comparer, si accordé, permet aux attributs et à leurs valeurs d'être lus. Par contre, s'il est refusé, il interdit toute forme d'accès aux attributs et à leurs valeurs.

Lire, si accordé, permet aux attributs et à leurs valeurs d'être lus et comparés.

Feuilleter, si accordé, permet la détection de l'existence d'une valeur d'attribut spécifique.

Les recommandations ne disent rien au sujet d'ajouter et d'enlever.

4.2.2. Etendues Spécifiques de Contrôle d'Accès.

Le DIT est partionné en sous-arbres appelés étendues administratives autonomes. Chacune de celles-ci se retrouve sous l'autorité administrative d'une unique organisation de gestion de domaine et peut être partionnée en sous-arbres appelés étendues administratives spécifiques dans le dessein d'aspects spécifiques de gestion.

Dans le cas du contrôle des accès, l'autorité administrative spécifique est une autorité de sécurité et l'étendue administrative spécifique est appelée étendue spécifique de contrôle d'accès. La racine de cette étendue est représentée par un point administratif de contrôle d'accès. Chacun de ces points correspond dans le DIT à une entrée administrative qui a une sous entrée attachée à celle-ci et qui contient l'information de contrôle des accès de base.

L'autorité administrative de sécurité peut accepter de partionner l'étendue spécifique de contrôle d'accès en sous-arbres appelés étendues administratives internes. Chacune de ces étendues est appelée étendue interne de contrôle d'accès. Comme précédemment, la racine de cette étendue est un point administratif représenté par une entrée administrative à laquelle

est attachée une sous-entrée contenant les informations de contrôle d'accès.

L'accès à une entrée donnée est contrôlé par la totalité des points administratifs de contrôle d'accès supérieurs (internes et spécifiques) jusqu'au premier point administratif de contrôle d'accès non-interne rencontré lorsqu'on remonte le DIT vers la racine de celui-ci.

4.2.3. Fonction de Décision de Contrôle d'accès - ACDF.

Cette section propose une description conceptuelle de la fonction de décision de contrôle d'accès (ACDF) pour un contrôle d'accès de base. Elle décrit comment les articles ACI (information de contrôle d'accès) sont traités afin de décider de donner ou refuser l'accès à une information.

Pour chaque décision de contrôle d'accès, les données sont :

- une ou plusieurs permissions.
- le nom distingué du demandeur.
- un identifiant associé au demandeur.
- le niveau d'authentification associé au demandeur.
- l'article protégé auquel la permission demandée est associée (entrée, attributs, valeurs).

Voyons maintenant la procédure de décision. Tout d'abord, pour l'entrée contenant l'article protégé considéré, on collecte toute l'information provenant des articles ACI qui s'applique à cette entrée. On place ensuite cette information dans un tableau du type suivant :

Pour chacune des lignes du tableau, les valeurs de colonnes sont :

- user qui spécifie une ou plusieurs classes d'utilisateurs.
- protected item qui spécifie l'article dont traite la ligne.
- permission qui est un ensemble de bits, avec chacun de ces bits correspondant à une catégorie de permission.
- grant/deny qui indique si les permissions sont acceptées ou refusées.
- unique identifier qui spécifie l'unique identifiant associé à l'utilisateur.
- precedence qui est un entier identifiant le niveau de précédence associé aux articles ACI.
- authentication level qui spécifie le niveau d'authentification requis nécessaire pour que la permission puisse s'appliquer.

Si toutes les lignes indiquent que l'accès est refusé ou qu'aucune ligne n'indique que l'accès est accepté, alors la permission est refusée et le processus de la fonction de décision du contrôle d'accès se termine.

Le fait que certaines permissions soient refusées peut impliquer que d'autres permissions le sont aussi automatiquement. C'est le cas dans les exemples suivants :

- si localiser est refusé, alors toutes les autres permissions le sont aussi.
- si comparer est refusé, alors feuilleter et lire le sont aussi.
- si lire est refusé, alors feuilleter l'est aussi.

On retire ensuite toutes les lignes qui ne sont pas pertinentes pour le demandeur en suivant la démarche suivante pour les lignes qui accordent la permission :

On retire les lignes où :

- le niveau d'authentification est insuffisant.
- le demandeur n'appartient pas à une certaine classe d'utilisateurs.

Cependant, on retire aussi les lignes qui refusent la permission si et seulement si :

- le niveau d'authentification rejoint ou dépasse celui spécifié dans la colonne.
- le demandeur ne fait pas partie d'une certaine classe d'utilisateurs.

Ensuite on retire les lignes qui ne sont pas pertinentes pour l'article protégé considéré et dont les bits de permission ne correspondent pas avec les permissions demandées.

La permission demandée repose sur une entrée, un attribut ou une valeur. On élimine donc les lignes ne traitant pas de ce sur quoi repose la permission.

On trie selon un ordre décroissant sur la précédence les lignes et on retire celles ayant une "petite" précédence. La permission est acceptée uniquement si les lignes restantes autorisent l'accès à l'article protégé.

5. Exemple de Contrôle d'Accès de Base. [X.501]

5.1. Introduction.

Le fragment de DIT représenté à la figure suivante est utilisé dans notre exemple. Il illustre une compagnie fictive : l'organisation Z. Sous l'élément country se trouve une entrée de classe d'objet organisation, qui a trois sous-arbres : Admin, R&D et Sales. Chacun de ceux-ci est représenté par une entrée de classe d'objet organizationalUnit, correspondant à des sites éloignés, sous lesquels apparaissent des entrées de classe d'objet organizationalPerson.

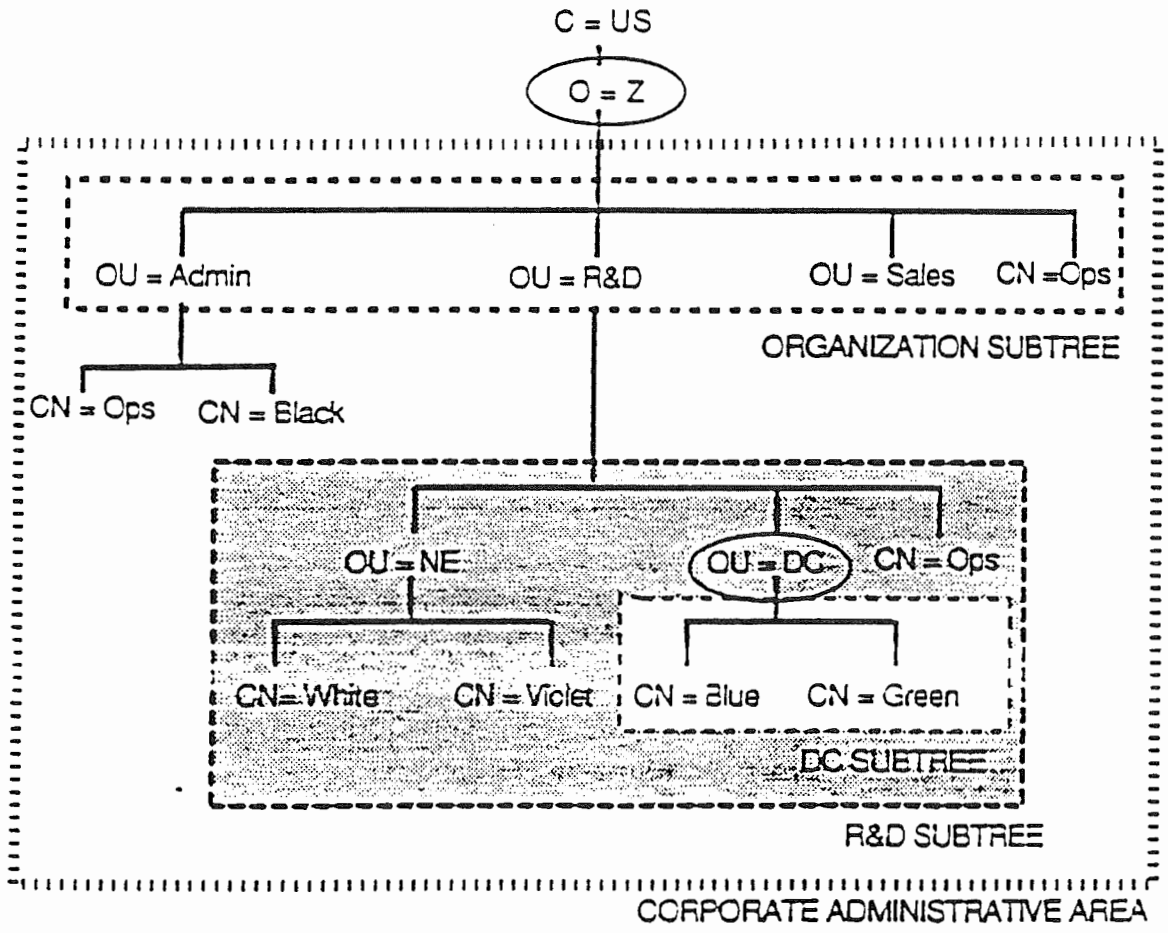


Fig 5.1 : Exemple de fragment de DIT.

Seulement quelques objets représentatifs de la classe organizationalPerson sont représentés. L'objet ayant le nom distingué {C=US, O=Z, OU=Admin, CN=Ops} est un objet de la classe groupOfUniqueNames; les valeurs de son attribut uniqueMember comprennent les administrateurs. Un des noms qu'il contient est {C=US, O=Z, OU=Admin, CN=Black}. Il existe deux autres groupes de

ce genre, un dont le nom distingué est {C=US, O=Z, OU=R&D, CN=Ops} et dont les membres sont responsables de la gestion des entrées du sous-arbre R&D. Un autre, dont le nom distingué est {C=US, O=Z, CN=Ops} a des membres responsables des objets de niveau organisationnel. L'utilisateur White dont le nom distingué est {C=US, O=Z, OU=R&D, OU=NE, CN=White} est un membre des deux derniers groupes.

Il y a deux points administratifs dans cet exemple. Ils sont indiqués par un cercle. Trois étendues ont été définies dans un but de contrôle d'accès. La Corporate Administrative Area est définie par le point administratif absolu {C=US, O=Z} et elle contient tous les subordonnés. Les sous-entrées de ce point administratif, non représentées sur le dessin, définissent les trois étendues spécifiques de contrôle d'accès. Une de celles-ci est l'étendue administrative entière. Une autre, dénommée Organization Subtree, comprend les objets d'unité organisationnelle de niveau supérieur. La dernière contient seulement le sous-arbre R&D. Une autre étendue administrative spécifique est définie à la sous-entrée du point d'accès {C=US, O=Z, OU=R&D, OU=DC}. Cette étendue comprend tous les subordonnés de ce point d'accès.

5.2. Politique.

L'entreprise veut renforcer la politique de contrôle d'accès suivante :

- les employés doivent être distingués du public. Les droits d'accès publiques seront limités de la façon suivante :

- les entrées peuvent être localisées par nom. La recherche par nom est permise. Par contre la recherche par numéro de téléphone par exemple n'est pas autorisée au public mais bien aux employés.

- les attributs publiques sont surname, telephoneNumber, les composants de postalAttributeSet et facimileTelephoneNumber.

- les employés ont le droit de lire et de rechercher pour la plupart des attributs de la plupart des entrées. Il existe cependant quelques exceptions comme userpassword qui ne peut être lu mais comparé.

- l'authentification simple est requise pour les accès des employés. L'authentification forte est requise pour les mises-à-jour.

- le sous-arbre DC a une autonomie complète pour établir sa propre politique.

- seuls les membres du groupe administratif {C=US, O=Z, OU=Admin, CN=Ops} peuvent ajouter, supprimer ou renommer des entrées utilisateurs.

- le groupe d'administrateurs {C=US, O=Z, OU=R&D, CN=Ops} est responsable des attributs utilisateurs du sous-arbre R&D, mais il ne peut pas ajouter, supprimer ou renommer des entrées.

- les personnes du sous-arbre R&D peuvent administrer eux mêmes des parties de leurs propres entrées en cas d'authentification forte. Ils peuvent ajouter des valeurs d'attribut à commonName, à facimileTelephoneNumber et à telephoneNumber, mais ils ne peuvent pas supprimer de valeur du numéro de téléphone fournie par l'administration.

- {C=US, O=Z, CN=Black} est un super-user autorisé à accéder à toutes les données et effectuer toutes les opérations nécessaires.

5.3. Attributs de Contrôle d'Accès.

Quatre niveaux de précédence sont utilisés : high, medium, low et default. Le point administratif absolu {C=US, O=Z} a trois sous-entrées definissant un contrôle d'accès de base. On remarquera que le sous-arbre DC, identifié comme étant une étendue administrative spécifique pour le contrôle d'accès, est exclu de la politique élaborée à ce point administratif.

5.3.1. Corporate Administrative Area.

Une sous-entrée de contrôle d'accès spécifie un sous-arbre dont l'étendue comprend la Corporate Administrative Area. Elle fournit les informations de contrôle d'accès (ACI) suivants, qui s'appliquent à toutes les entrées, exceptées celles du sous-arbre DC.

Identification:	"Public Access"
Precedence:	default
User Class:	allUsers
Authentication Level:	none
Protected Items:	*(A1)* thisEntry: grant **browse**
	(A2) allAttributeValues { **postalAttributeSet, telephoneNumber, aliasedObjectName, facimileTelephoneNumber** }: grant **read**
	(A3) allAttributeValues { **commonName** }: grant **browse**

Identification:	"Employee Access"
Precedence:	default
User Class:	subtree @ {C=US, O=Z}
Authentication Level:	simple
Protected Item:	*(B1)* allAttributeTypesAndValues: grant **browse**

Identification:	"Password protection"
Precedence:	medium
User Class:	allUsers
Authentication Level:	none
Protected Items:	*(C1)* allAttributeValues { **userPassword** }: deny **read** grant **compare**

Identification:	"Minimal Leaf Entry Administration"
Precedence:	low
User Class:	userGroup {C=US, O=Z, OU=Admin, CN=Ops}
Authentication Level:	strong
Protected Items:	*(D1)* entry: grant **add, remove, rename**
	(D2) allAttributeValues { **commonName, surname** } grant **add, remove**

Identification:	"Black is superuser"
Precedence:	high
UserClass:	user {C=US, O=Z, OU=Admin, CN=Black} uniqueIdentifier = 12345
Authentication Level:	strong
Protected Items:	*(E1)* entry grant **browse** grant **add, remove, rename, modify**
	(E2) allAttributeTypesAndValues : grant **browse, add, remove**

5.3.2. Sous-arbre Organisation.

Une autre sous-entrée comprend seulement les subordonnés immédiats du point administratif absolu : le sous-arbre Organisation. Les ACI sont les suivants :

Identification:	"Organizational Management"
Precedence:	medium
User Class:	userGroup {C=US, O=Z, CN=Ops}
Authentication Level:	strong
Protected Item:	*(F1)* entry grant add, remove, rename, modify
	(F2) allAttributeTypesAndValues: grant add, remove

Identification:	"Exclusive Organizational Management"
Precedence:	low
User Class:	allUsers
Authentication Level:	none
Protected Items:	*(G1)* entry: deny add, remove, rename, modify
	(G2) allAttributeTypesAndValues: deny add, remove

5.3.3. Sous-arbre R&D.

La dernière sous-entrée définit les ACI pour tous les subordonnés de {C=US, O=Z, OU=R&D}. On remarquera que les utilisateurs ne peuvent pas installer des politiques de contrôle d'accès sur leurs propres entrées.

Identification:	"User object self administration"
Precedence:	low
User Class:	thisEntry
Authentication Level:	strong
Protected Item:	*(H1)* entry grant **modify**
	(H2) allAttributeValues:{ **telephoneNumber, commonName, facsimileTelephoneNumber** }: **grant add, remove**

Identification:	"R&D administration"
Precedence:	low
User Class:	userGroup { C=US, O=Z, OU=R&D, CN=Ops }
Authentication Level:	strong
Protected Items:	*(I1)* entry: grant **modify**
	(I2) allAttributeTypesAndValues: grant add, remove

Identification:	"R&D Operations Group"
Entry:	{ C=US, O=Z, OU=R&D, CN=Ops }
Precedence:	medium
User Class:	allUsers

Authentication Level:	none
Protected Item:	*(J1)* entry deny **add, remove, modify**
	(J2) allAttributeTypesAndValues: deny **browse, add, remove**

Identification:	"Administration Supplied Telephone Number"
Entry:	(each entry)
Precedence:	medium
User Class:	entry
Authentication Level:	none
Protected Item:	*(K1)* attributeValue:{ **telephoneNumber** = (supplied by company)} deny remove

5.4. Exemple de Fonction de Décision de Contrôle d'Accès - ACDF

Les essais d'accès suivants seront effectués :

- un membre du public, dont le nom distingué est {C=GB, O=X, CN=Smith}, essaie de lire le numéro de téléphone de l'utilisateur white.
- le même demandeur essaie de rechercher toutes les valeurs d'attributs des utilisateurs situés dans {C=US, O=Z, OU=NE}.
- l'utilisateur Violet essaie de modifier des attributs et d'ajouter un numéro de téléphone à sa propre entrée.
- l'utilisateur Green recherche un utilisateur avec un numéro de téléphone particulier.
- l'utilisateur Green essaie de comparer une valeur de mot de passe pour l'utilisateur White.
- l'utilisateur White essaie de renommer Violet en Puce.

5.4.1. Table.

La table suivante est constituée des ACI qui s'appliquent au sous-arbre tout entier.

Les permissions sont représentées de la façon suivante :

entrée :	locate (situer)	L	attribut:	compare	c
	browse (feuilleter)	B		read (lire)	r
	add (ajouter)	A		browse	b
	remove (supprimer)	D		add	a
	rename (renommer)	N		remove	d
	modify (modifier)	M			

Les permissions sont préfixées d'un "+" quand elles sont acceptées et d'un "-" quand elles sont refusées. Le vecteur entier des permissions est indiqué, avec une barre signifiant l'absence d'une permission particulière.

Id	User	Item	Perm	Prec	Authen	UniqueID
A1	all users	entry	+ -B----	default	none	
A2	all users	postalAttributes values	+ -r---	default	none	
A2	all users	alias name values	+ -r---	default	none	
A2	all users	fax number values	+ -r---	default	none	
A2	all users	telephone number values	+ -r---	default	none	
A3	all users	common name values	+ --b--	default	none	
B1	subtree Z	all attribute values	- --b--	default	simple	
C1	allUsers	password values	- -rb--	medium	none	
C1	allUsers	password values	+ c----	medium	none	
D1	Adm Ops	entry	+ --ADN-	low	strong	
D2	Adm Ops	common name values	+ ---ad	low	strong	
D2	Adm Ops	surname values	+ ---ad	low	strong	
E1	Black	entry	+ -BADNM	high	strong	12345
E2	Black	all attribute values	+ --bad	high	strong	12345

Pour le sous-arbre organisation, la table est la suivante :

F1	Ops group	entry	+ --ADNM	medium	strong
F2	Ops group	all attribute values	+ ---ad	medium	strong
G1	all users	entry	- --ADNM	low	none
G2	all users	all attribute values	- ---ad	low	none

Pour le sous-arbre R&D, la table est la suivante :

H1	this entry	entry	+ -----M	low	strong
H2	this entry	telephone number values	+ ---ad	low	strong
H2	this entry	common name values	+ ---ad	low	strong
H2	this entry	fax number values	+ ---ad	low	strong
I1	R&D Ops	entry	+ -----M	low	strong
I2	R&D Ops	all attribute values	+ ---ad	low	strong

5.4.2. Accès publique en lecture.

Un membre du public, dont le nom distingué est {C=GB, O=X, CN=Smith}, essaie de lire le numéro de téléphone de l'utilisateur white.

L'opération se déroule de la manière suivante :

Pas 1 :

On détermine si Smith a la permission d'accéder au niveau entrée en se rapportant à l'input de la fonction ACDF suivant :

- permission requise : R ou B.
- demandeur : {C=GB, O=X, CN=Smith}
- identifiant unique : néant
- niveau d'authentification : néant
- article protégé : entrée {C=US, O=Z, OU=R&D, OU=NE, CN=White}

L'entrée demandée se trouve sous le couvert du sous-arbre administratif et R&D. La table est la suivante :

A1	all users	entry	+ -B----	default	none
A2	all users	postalAttributes values	+ -r----	default	none
A2	all users	alias name values	- -r----	default	none
A2	all users	fax number values	- -r----	default	none
A2	all users	telephone number values	+ -r----	default	none
A3	all users	common name values	+ --b---	default	none
C1	allUsers	password values	- -rb---	medium	none
C1	allUsers	password values	+ c-----	medium	none

Seulement la première ligne est utile pour cet article protégé. La permission demandée est accordée.

Pas 2 :

On détermine si Smith a la permission d'accéder au niveau attribut. Pour cet accès, l'input de la fonction ACDF est le suivant :

- permission requise : r ou b
- demandeur : {C=GB, O=X, CN=Smith}
- identifiant unique : néant
- niveau d'authentification : néant
- article protégé : telephoneNumber

La même table est de rigueur. La dernière ligne concernant l'accès aux valeurs d'attribut est la seule utile et la permission est accordée.

5.4.3. Accès Publique en Recherche.

Notre Smith essaie maintenant de rechercher toutes les valeurs de tous les attributs pour tous les utilisateurs subordonnés à l'objet de base {C=US, O=Z, OU=R&D, OU=NE}. Les actions suivantes sont effectuées :

Pour chaque entrée située dans l'étendue de recherche, on détermine si la permission de niveau entrée (B) et la permission de niveau attribut (r) est accordée. La permission b n'est pas obligatoire ici puisque aucune valeur d'attribut n'apparaît dans le filtre de recherche.

Pas 1 :

Pour la première entrée située dans l'étendue de recherche, on détermine si Smith a la permission de niveau entrée grâce à l'input de la fonction ACDF suivant :

- permission requise : B
- demandeur : {C=GB, O=X, CN=Smith}
- identifiant unique : néant
- niveau d'authentification : néant

- article protégé : entrée {C=US, O=Z, OU=R&D, OU=NE}

L'entrée demandée se trouve dans le sous-arbre administratif et R&D. La table est donc la suivante :

A1	all users	entry	+ -e----	default	none
A2	all users	postalAttributes values	+ -r---	default	none
A2	all users	alias name values	+ -r---	default	none
A2	all users	fax number values	+ -r---	default	none
A2	all users	telephone number values	+ -r---	default	none
A3	all users	common name values	+ --b--	default	none
B1	subtree Z	all attribute values	+ --b--	default	simple
C1	allUsers	password values	- -rb--	medium	none
C1	allUsers	password values	+ c----	medium	none

Seulement la ligne A1 est utile pour cet article protégé. La permission requise est B et elle est accordée.

Pas 2 :

On détermine si Smith a la permission de niveau attribut. Pour cet accès, l'input de la fonction ACDF est le suivant :

- permission requise : r ou b
- demandeur : {C=GB, O=X, CN=Smith}
- identifiant unique : néant
- niveau d'authentification : néant
- article protégé : tous les attributs et leurs valeurs

On réutilise la même table. La dernière ligne (B1) est la seule utile. Les seules permissions accordées sont celles se rapportant au numèro de fax, de téléphone et aux attributs postaux.

5.4.4. Modification de Valeur d'Attributs.

Un utilisateur dont le nom distingué {C=US, O=Z, OU=R&D, OU=NE, CN=Violet} essaie d'ajouter une valeur de numéro de téléphone à sa propre entrée.

Pas 1 :

On détermine si Violet a la permission de niveau entrée en accord avec l'input de la fonction ACDF :

- permission requise : M
- demandeur : {C=US, O=Z, OU=R&D, OU=NE, CN=Violet}

- identifiant unique : néant
- niveau d'authentification : forte
- article protégé : {C=US, O=Z, OU=R&D, OU=NE, CN=Violet}

L'entrée en question se trouve dans le sous-arbre administratif et R&D. La table est donc la suivante :

A1	all users	entry	+ -B----	default	none
A2	all users	postalAttributes values	+ -r---	default	none
A2	all users	alias name values	+ -r---	default	none
A2	all users	fax number values	+ -r---	default	none
A2	all users	telephone number values	+ -r---	default	none
A3	all users	common name values	+ --b--	default	none
B1	subtree Z	all attribute values	+ --b--	default	simple
C1	allUsers	password values	- -rb--	medium	none
C1	allUsers	password values	+ c----	medium	none
H1	this entry	entry	+ -----M	low	strong
H2	this entry	telephone number values	+ ---ad	low	strong
H2	this entry	common name values	+ ---ad	low	strong
H2	this entry	fax number values	+ ---ad	low	strong

On retire les lignes inutiles pour cet article protégé et on obtient la table suivante :

A1	all users	entry	+ -B----	default	none
H1	this entry	entry	+ -----M	low	strong

La permission de modification est accordée par la ligne H1.

Pas 2 :

On détermine si Violet est autorisé à modifier la valeur de l'attribut en question. L'input de la fonction ACDF est :

- permission requise : a
- demandeur : {C=US, O=Z, OU=R&D, OU=NE, CN=Violet}
- identifiant unique : néant
- niveau d'authentification : forte
- article protégé : une valeur pour telephoneNumber

On élimine les lignes inutiles pour l'article protégé.

A2	all users	telephone number values	+ -r---	default	none
B1	subtree Z	all attribute values	+ --b--	default	simple
H2	this entry	telephone number values	+ ---ad	low	strong

La ligne H2 accorde la permission.

5.4.5. Recherche par Valeur d'Attribut.

Un utilisateur dont le nom distingué est {C=US, O=Z, OU=R&D, OU=DC, CN=Green} essaie de rechercher les noms des objets se trouvant dans le sous-arbre R&D et ayant une valeur de numéro de téléphone particulière.

Pour chaque entrée de l'étendue de recherche, on détermine si les permissions de niveau entrée (B) et de niveau attribut (b) sont accordée.

Pas 1 :

Pour chaque entrée, on détermine si Green a la permission d'accès de niveau entrée en accord avec l'input de la fonction ACDF :

- permission requise : B
- demandeur : {C=US, O=Z, OU=R&D, OU=DC, CN=Green}
- identifiant unique : néant
- niveau d'authentification : simple
- article protégé : {C=US, O=Z, OU=R&D, OU=NE, CN=White}

L'entrée requise se trouve dans le sous-arbre administratif et R&D. La table obtenue est la suivante :

A1	all users	entry	+ -B----	default	none
A2	all users	postalAttributes values	+ -r---	default	none
A2	all users	alias name values	+ -r---	default	none
A2	all users	fax number values	+ -r---	default	none
A2	all users	telephone number values	+ -r---	default	none
A3	all users	common name values	+ --b--	default	none
B1	subtree Z	all attribute values	+ --b--	default	simple
C1	allUsers	password values	- -rb--	medium	none
C1	allUsers	password values	+ c----	medium	none
H1	this entry	entry	+ -----M	low	strong
H2	this entry	telephone number values	+ ---ad	low	strong
H2	this entry	common name values	+ ---ad	low	strong
H2	this entry	fax number values	- ---ad	low	strong
K1	this entry	telephone number "n"	+ ----d	low	strong

La ligne A1 est la seule utile pour l'article protégé et la permission est accordée.

Pas 2 :

- permission requise : b
- demandeur : {C=US, O=Z, OU=R&D, OU=DC, CN=Green}
- identifiant unique : néant
- niveau d'authentification : simple
- article protégé : n'importe quelle valeur d'attribut telephoneNumber

La table est la suivante :

A2	all users	telephone number values	+ -r---	default	none
H2	this entry	telephone number values	- ---ad	low	none
B1	subtree Z	all attribute values	+ --b--	default	simple
K1	this entry	telephone number "n"	- ----d	medium	none

La ligne B1 accorde la permission requise.

5.4.6. Comparaison de Valeur.

Un utilisateur {C=US, O=Z, OU=R&D, OU=DC, CN=Green} essaie de comparer un mot de passe pour l'utilisateur White.

Pas 1:

On détermine l'accès de niveau entrée :

- permission requise : L ou B
- demandeur : {C=US, O=Z, OU=R&D, OU=DC, CN=Green}
- identifiant unique : néant
- niveau d'authentification : simple
- article protégé : {C=US, O=Z, OU=R&D, OU=NE, CN=White}

L'entrée se trouve dans le sous-arbre administratif et R&D. La table est donc la suivante :

A1	all users	entry	+ -B----	default	none
A2	all users	postalAttributes values	+ -r---	default	none
A2	all users	alias name values	+ -r---	default	none
A2	all users	fax number values	+ -r---	default	none
A2	all users	telephone number values	+ -r---	default	none
A3	all users	common name values	+ --b--	default	none
B1	subtree Z	all attribute values	+ --b--	default	simple
C1	allUsers	password values	- -rb--	medium	none
C1	allUsers	password values	- c----	medium	none

De toutes celles-ci, seule la première est utile et la permission requise est accordée.

Pas 2 :

On détermine si Green a la permission d'accès de niveau attribut. L'input de la fonction ACDF est le suivant :

- permission requise : c,r ou b
- demandeur : {C=US, O=Z, OU=R&D, OU=DC, CN=Green}
- identifiant unique : néant
- niveau d'authentification : simple
- article protégé : UserPassword

Nous obtenons alors la table suivante, ordonnée suivant les niveaux de préférence.

C1	all users	password values	- --rb--	medium	none
C1	all users	password values	+ c----	medium	none
B1	subtree Z	all attribute values	+ --b--	default	simple

Les permissions b et r sont refusées. La permission c est accordée.

5.4.7. Modification de RDN.

L'utilisateur White essaie de renommer l'entrée Violet en Puce. White est un membre de l'organisation Ops et du groupe Ops R&D. Il est fortement authentifié, avec un identifiant unique.

Premièrement on détermine l'accès de niveau entrée pour le RDN courant. L'input de la fonction ACDF est le suivant :

- permission requise : N
- demandeur : {C=US, O=Z, OU=R&D, OU=NE, CN=White}
- identifiant unique : fourni
- niveau d'authentification : forte
- article protégé : {C=US, O=Z, OU=R&D, OU=NE, CN=Violet}

La table obtenue est la suivante car l'entrée en question appartient au sous-arbre administratif et R&D.

A1	all users	entry	+ -B----	default	none
A2	all users	postalAttributes values	+ -r----	default	none
A2	all users	alias name values	+ -r----	default	none
A2	all users	fax number values	+ -r----	default	none
A2	all users	telephone number values	- -r----	default	none
A3	all users	common name values	- --b---	default	none
B1	subtree Z	all attribute values	- --b---	default	simple
C1	allUsers	password values	- -rb---	medium	none
C1	allUsers	password values	- c-----	medium	none
I1	R&D Ops	entry	- -----M	low	strong
I2	R&D Ops	all attribute values	- ---ad	low	strong

De toutes les lignes, seules A1 et I1 sont utiles et la permission n'est pas accordée.

6. Le Service Abstrait du Directory - X.511.

6.0. Introduction.

Cette section propose une revue des recommandations X.511 version 1992. Nous ne parlerons ici que des modifications à l'ancienne recommandation. Ces modifications se rapportent au service de sécurité lié aux opérations abstraites du Directory.

6.1. Opérations de Connexion et de Déconnexion.

Un argument est ajouté aux arguments de l'opération de connexion (Bind). Il s'appelle accessControlData et transporte les informations de contrôle d'accès.

6.2. Opérations de Lecture du Directory.

Un argument, dénommé basic-access-control, est ajouté à l'ensemble des arguments de toutes les opérations de lecture.

6.2.1. Read Operation.

Si le basic-access-control est en action pour l'entrée en cours de lecture, on applique le mécanisme suivant :

- la permission situer ou feuilleter est requise pour l'entrée identifiée par les arguments de l'opération read. Ceci est aussi valable quand l'entrée est une entrée alias et doit être déréférenciée. Si la permission n'est pas accordée, l'opération échoue.
- si l'entrée est un alias, et que la déréférenciation est requise, la permission situer ou feuilleter est requise avant que la déréférenciation puisse avoir lieu. Si la permission n'est pas accordée, l'opération échoue.
- si dans les arguments de l'opération read, on demande que les types d'attributs soient retournés, il faut que pour chacun de ceux-ci, la permission situer ou feuilleter soit accordée sinon l'opération échoue.
- si les types d'attributs et les valeurs de ceux-ci sont demandés, on applique le même principe que précédemment.

Les figures suivantes illustrent ce mécanisme.

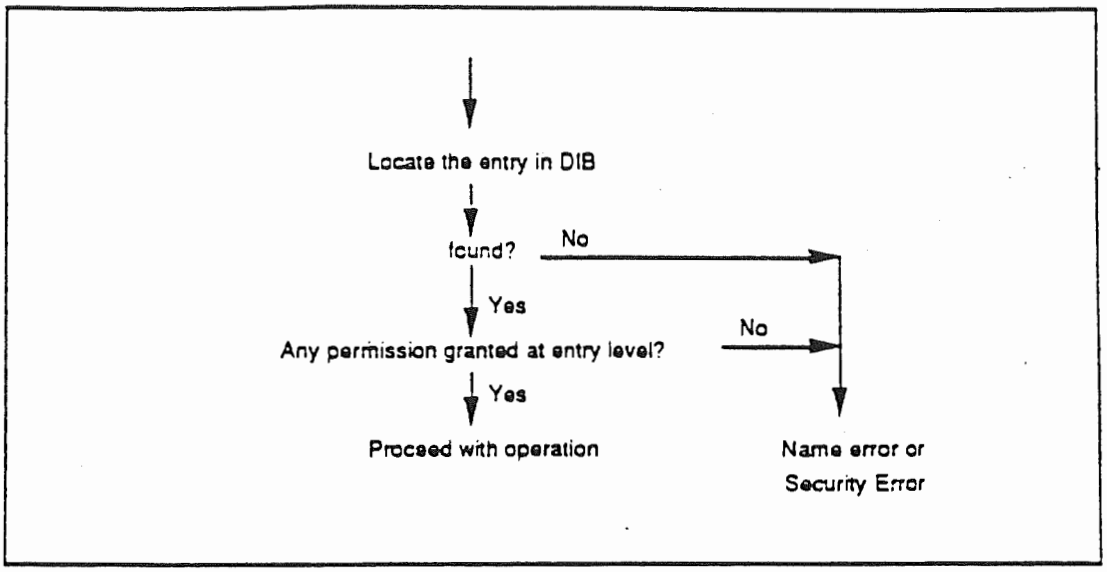


Fig 6. : Entry visibility.

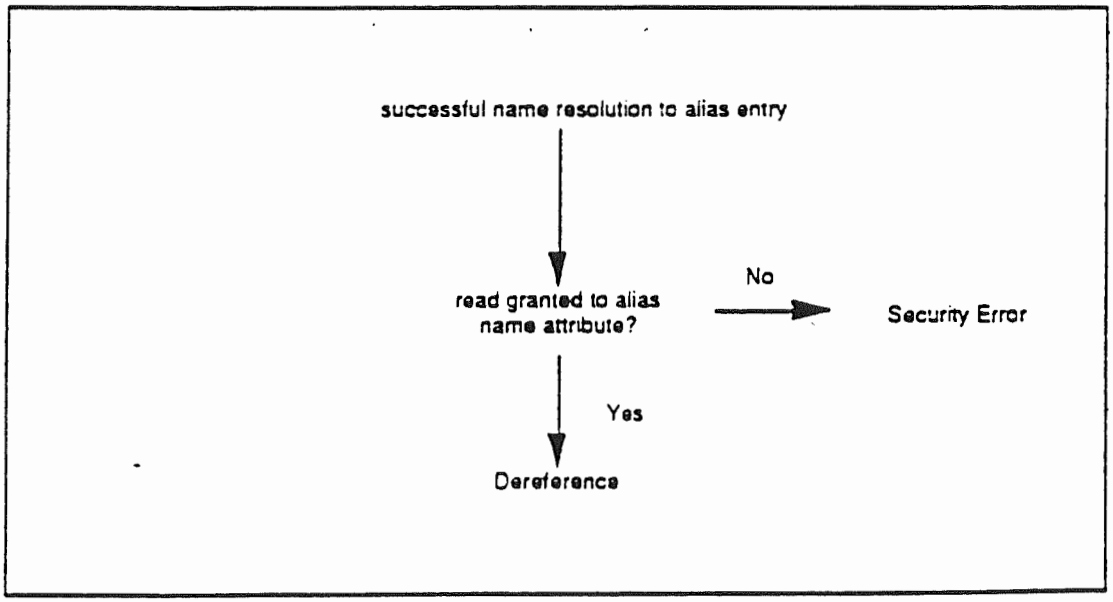


Fig 6.2. : Alias dereferencing.

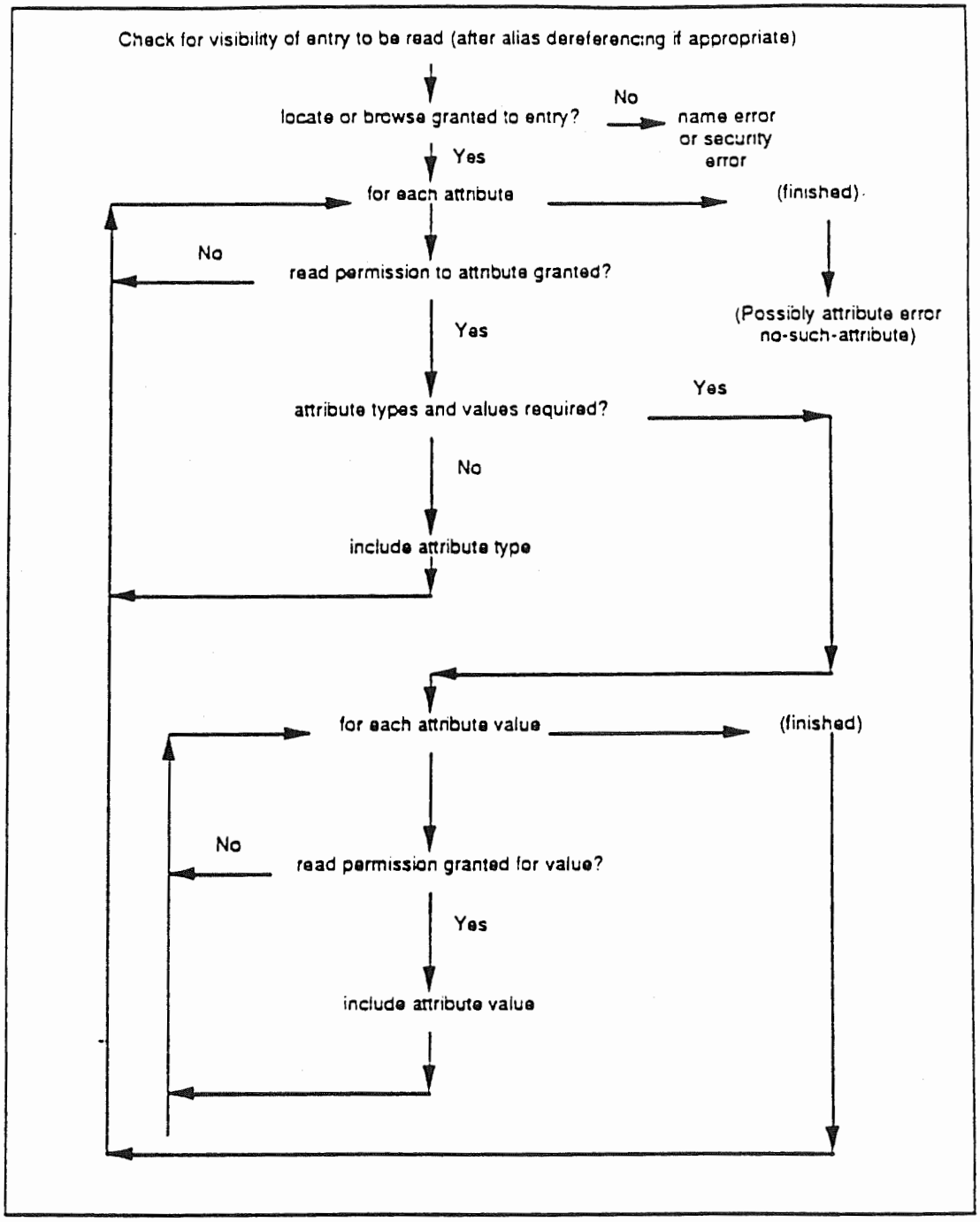


Fig 6.3. : L'opération read.

Si l'opération échoue, une erreur est retournée :

- NameError en cas de problème noSuchObject.
- NameError en cas de problème aliasDereferencingProblem.
- SecurityError en cas de problème insufficientAccessRights.

6.2.2. Compare Operation.

Si le contrôle des accès de base (basic-access-control)est en action pour l'entrée en cours de comparaison, le mécanisme suivant est d'application :

- la permission situer ou feuilleter est requise pour l'entrée identifiée par les arguments de l'opération compare. Ceci s'applique aussi quand l'entrée est un alias. Si la permission n'est pas accordée, l'opération échoue.
- si l'entrée est un alias, et que la déréférenciation est requise, la permission situer ou feuilleter est requise avant que la déréférenciation puisse avoir lieu. Si la permission n'est pas accordée, l'opération échoue.
- la permission comparer, lire ou feuilleter est requise pour l'attribut en cours de comparaison, sinon l'opération échoue.
- la permission comparer, lire ou feuilleter est requise pour chaque valeur d'attribut en cours de comparaison, sinon l'opération échoue.

Le diagramme suivant illustre ce mécanisme.

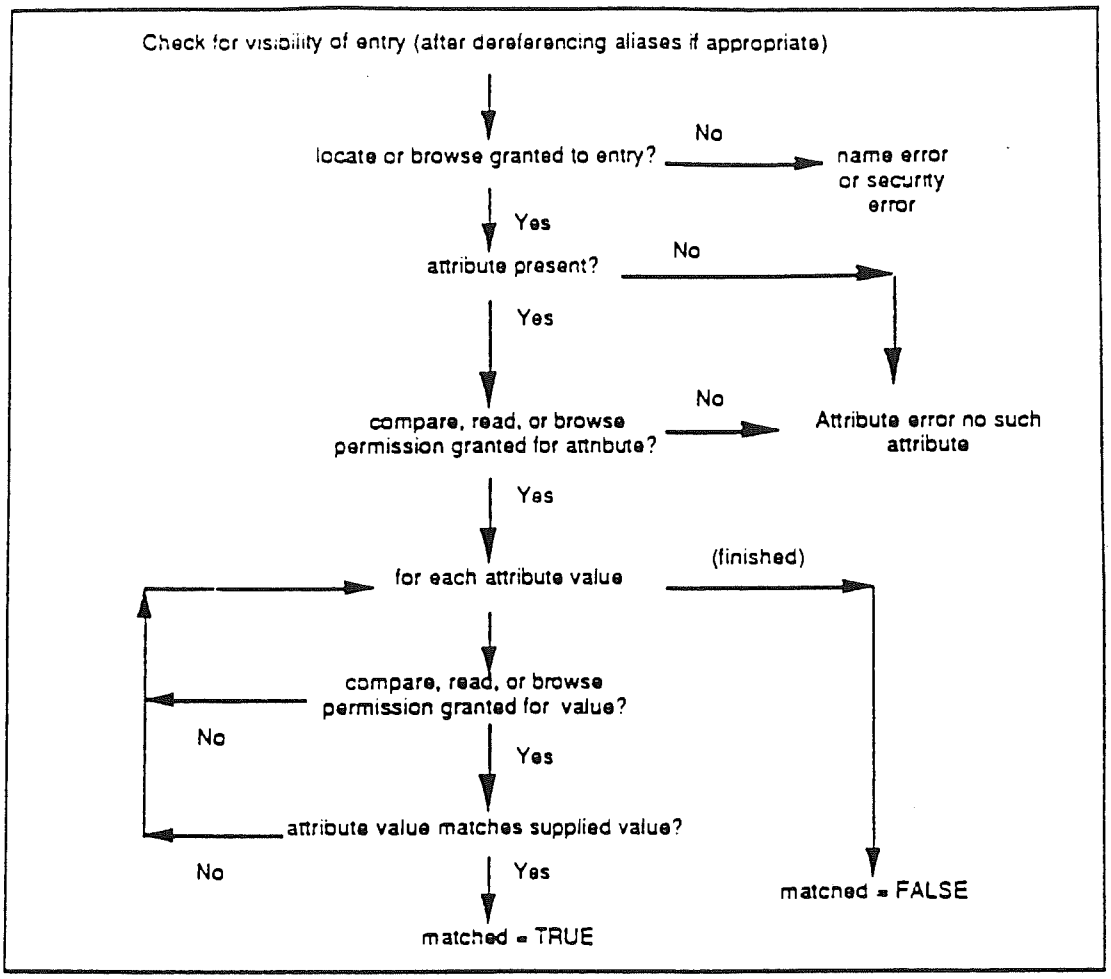


Fig 6.4. : L'opération compare.

Si l'opération échoue, une erreur est retournée :

- NameError en cas de problème noSuchObject.
- NameError en cas de problème aliasDereferencingProblem.
- SecurityError en cas de problème insufficientAccessRights.
- AttributeError en cas de problème noSuchAttributeOrValue.

6.3. Opérations de Recherche du Directory.

Un argument, dénommé basic-access-control, est ajouté à l'ensemble des arguments de toutes les opérations de recherche.

6.3.1. List Operation.

Si le contrôle des accès de base est en action, le mécanisme suivant est d'application pour l'entrée dont les subordonnés sont en cours d'énumération (listing) :

- aucune permission n'est requise pour l'entrée identifiée par les arguments de l'opération list sauf si la déréférenciation est de rigueur. Dans ce cas la permission situer ou feuilleter est requise. Si cette permission n'est pas accordée, l'opération échoue.
- si l'entrée identifiée par les arguments de l'opération list est une entrée de type alias et que la déréférenciation est requise, la permission lire ou feuilleter est de rigueur avant que la déréférenciation puisse prendre place. Si la permission n'est pas accordée, l'opération échoue.
- pour chacun des subordonnés immédiats pour lequel le RDN doit être retourné, la permission feuilleter est requise pour cette entrée. Les entrées pour lesquelles la permission est refusée sont purement et simplement ignorées.

Le diagramme suivant illustre ce mécanisme.

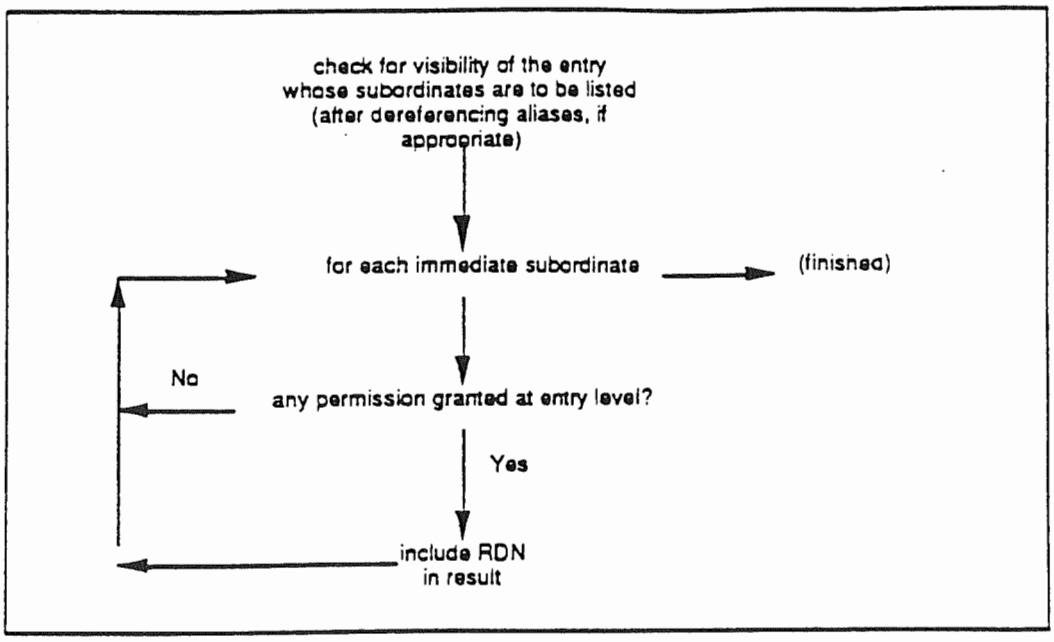


Fig 6.5. : L'opération list.

Si l'opération échoue, une erreur est retournée :

- NameError en cas de problème noSuchObject.
- NameError en cas de problème aliasDereferencingProblem.
- SecurityError en cas de problème insufficientAccessRights.

6.3.2. Search Operation.

Si le contrôle des accès de base est en action pour la partie duDIT en cours de recherche, le mécanisme suivant est d'application :

- aucune permission n'est requise pour l'entrée identifiée par les arguments de l'opération search sauf si la déréférenciation est de rigueur. Dans ce cas la permission situer ou feuilleter est requise. Si cette permission n'est pas accordée, l'opération échoue.
- si l'entrée identifiée par les arguments de l'opération search est une entrée de type alias et que la déréférenciation est requise, la permission lire ou feuilleter est de rigueur avant que la déréférenciation puisse prendre place. Si la permission n'est pas accordée, l'opération échoue.
- pour chacune des entrées situées dans l'étendue de recherche pouvant être prise en considération, la permission feuilleter est requise. Les entrées pour lesquelles cette permission est refusée sont ignorées.

Le filtre de recherche est appliqué à chacune des entrées restantes et on applique alors le mécanisme suivant :

- pour chaque élément du filtre qui spécifie un type d' attribut, la permission feuilleter est requise avant que cet élément de filtre puisse être évalué comme vrai ou faux. Un élément du filtre pour lequel cette permission n'est pas accordée est indéfini.

- pour chaque élément du filtre qui spécifie en plus une valeur d'attribut, la permission feuilleter est requise pour chaque valeur d'attribut stockée qui doit être considérée dans un but d'appariement. Si la permission est acceptée et que la valeur de l'attribut s'accorde avec le filtre, l'élément du filtre est évalué à vrai.

Le filtre est évalué et les entrées sont acceptées ou refusées. Si elles sont acceptées, l'information retournée est la suivante:

- si le filtre stipule que seuls les types d'attributs doivent être retournés, alors pour chacun de ceux-ci, le permission lire ou feuilleter est requise.

- si le filtre stipule que les types et valeurs d'attributs doivent être retournés, alors pour chacun de ceux-ci, la permission lire ou feuilleter est requise.

Le diagramme suivant illustre ce mécanisme.

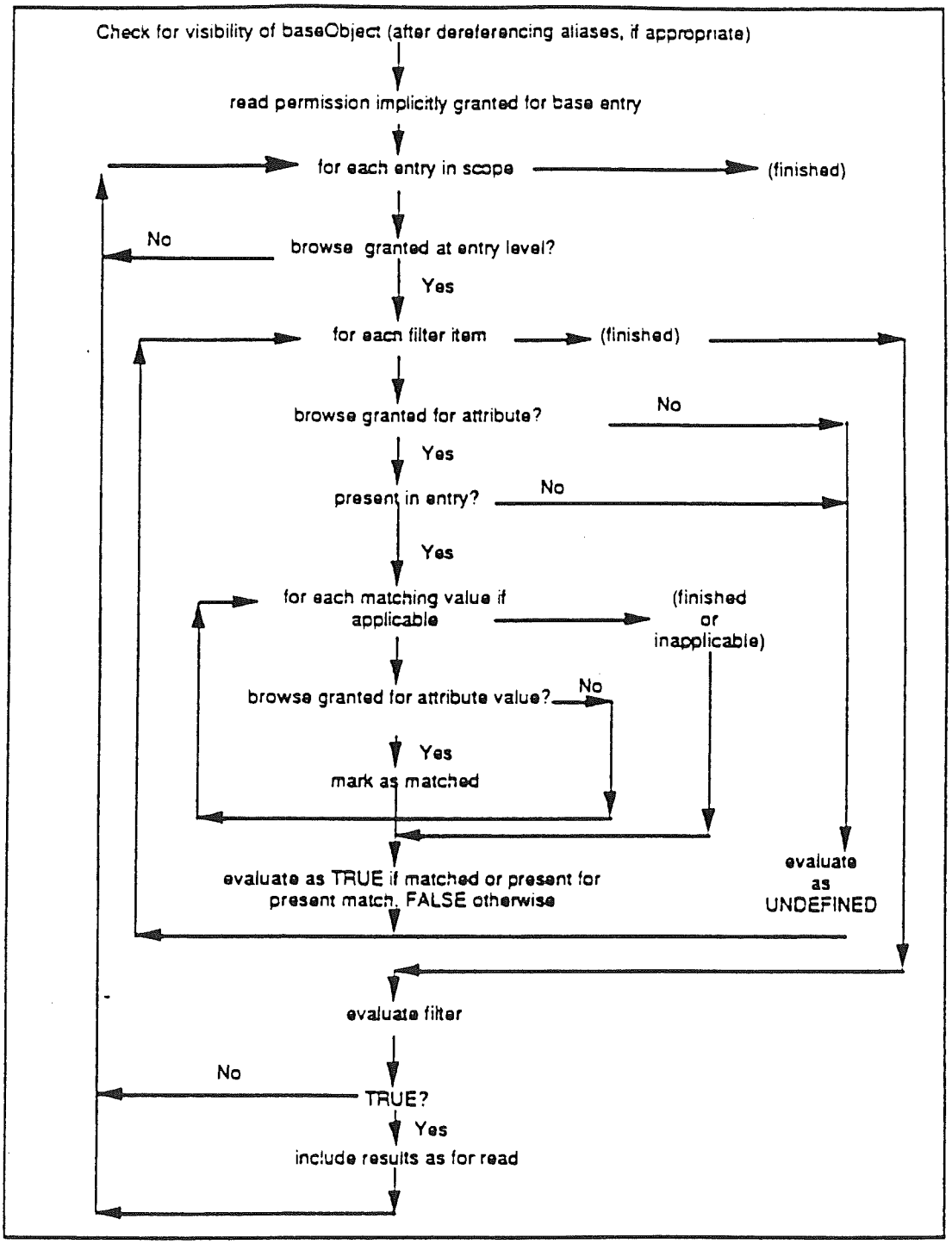


Fig 6.6. : L'opération search.

Si l'opération échoue, une erreur est retournée :
- NameError en cas de problème noSuchObject.
- NameError en cas de problème aliasDereferencingProblem.
- SecurityError en cas de problème insufficientAccessRights.

6.4. Opérations de Modification du Directory.

6.4.1. Add Entry Operation.

Si le contrôle des accès de base est en application pour l'entrée en cours d'ajout, le mécanisme suivant est d'application :

- la permission situer ou feuilleter est requise pour le supérieur immédiat de l'entrée identifiée par les arguments de la fonction add entry. Si la permission est refusée, l'opération échoue.
- si une entrée de même nom que celle qu'on désire ajouter existe déjà, l'opération échoue.
- la permission ajouter est requise pour la nouvelle entrée sinon l'opération échoue.
- pour chaque type et valeur d'attribut à ajouter, la permission ajouter est requise sinon l'opération échoue.

Le diagramme suivant illustre ce mécanisme.

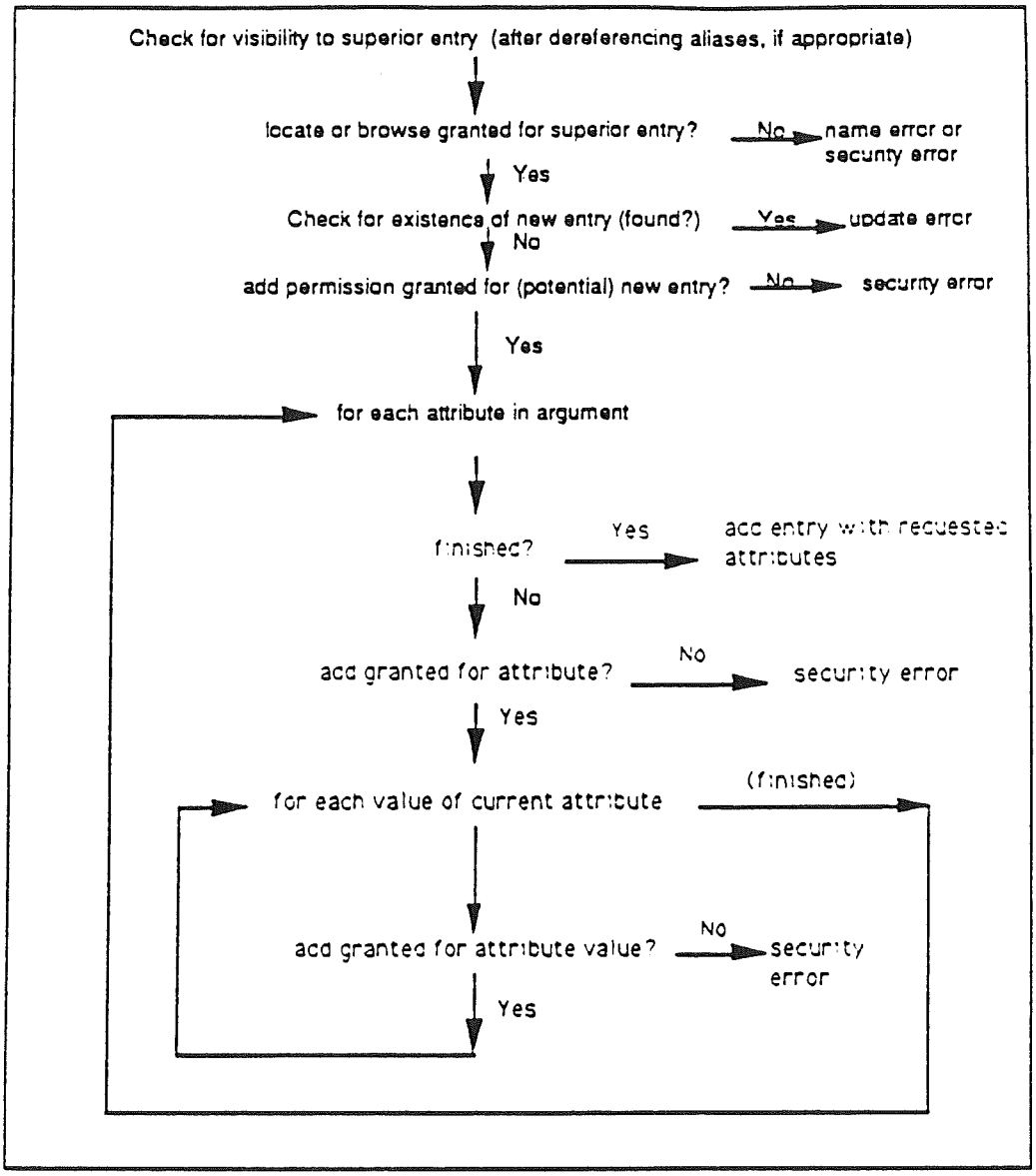


Fig 6.7. : L'opération add entry.

Si l'opération échoue, une erreur est retournée :

- NameError en cas de problème noSuchObject.
- SecurityError en cas de problème insufficientAccessRights.

6.4.2. Remove Entry Operation.

Si le contrôle des accès de base est en application pour l'entrée en cours d'enlèvement, le mécanisme suivant est d'application :

- la permission retirer est requise pour l'entrée en cours d'enlèvement. Si cette permission est refusée, l'opération échoue.
- pour chaque type et valeur d'attribut présent dans l'entrée, la permission retirer est requise, sinon l'opération échoue.

Le diagramme suivant illustre ce mécanisme.

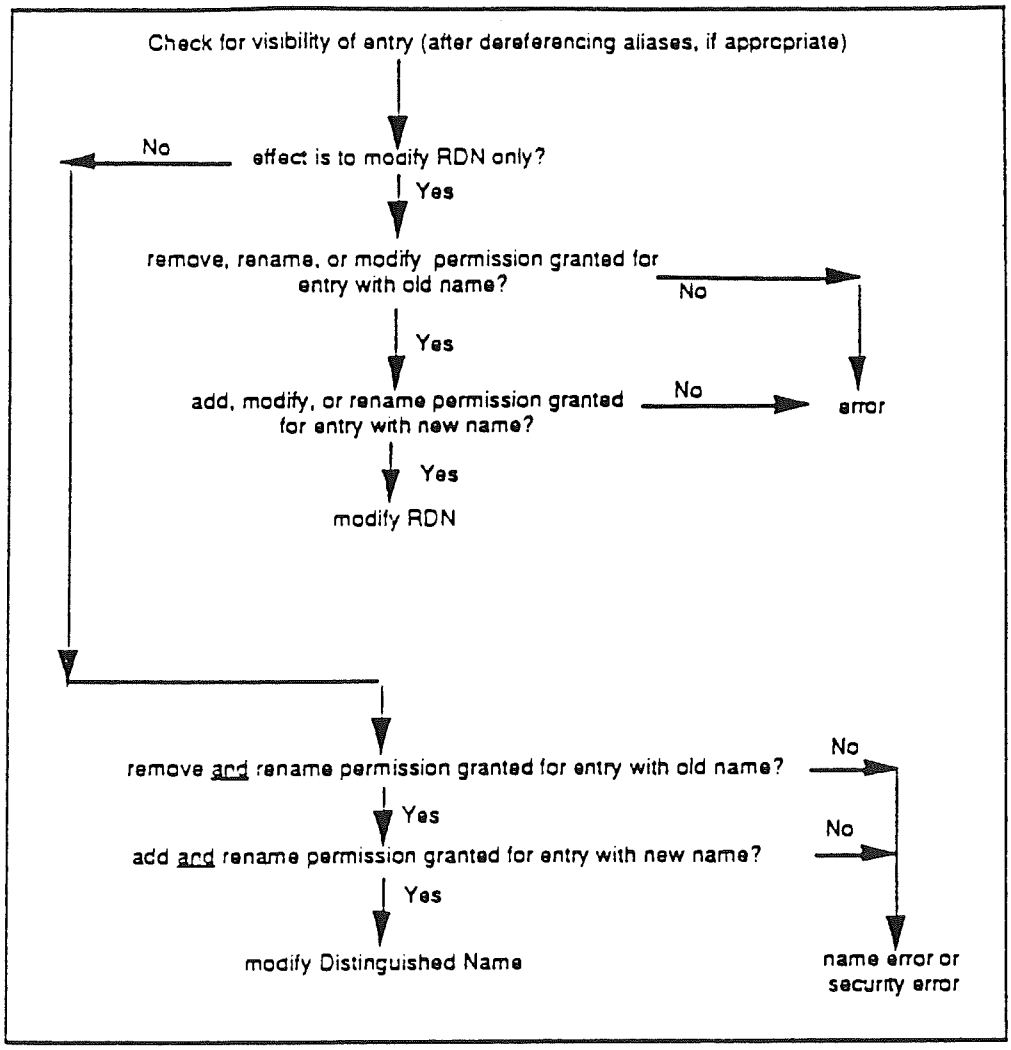


Fig 6.8. : L'opération remove entry.

Si l'opération échoue, une erreur est retournée :

- NameError en cas de problème noSuchObject.
- SecurityError en cas de problème insufficientAccessRights.

6.4.3. Modify Entry Operation.

Si le contrôle des accès de base est en application pour l'entrée en cours de modification,le mécanisme suivant est d'application :

- la permission modifier est requise pour l'entrée concernée, sinon l'opération échoue.
- pour chaque élément à modifier dans l'entrée, les permissions suivantes sont requises :

- la permission ajouter pour chaque type et valeur d'attribut à ajouter.
- la permission retirer pour chaque type et valeur d'attribut à supprimer.
- la permission ajouter pour chaque valeur d'attribut à ajouter.
- la permission retirer pour chaque valeur d'attribut à retirer.

Si une permission est absente, l'opération échoue.

Le diagramme suivant illustre ce mécanisme.

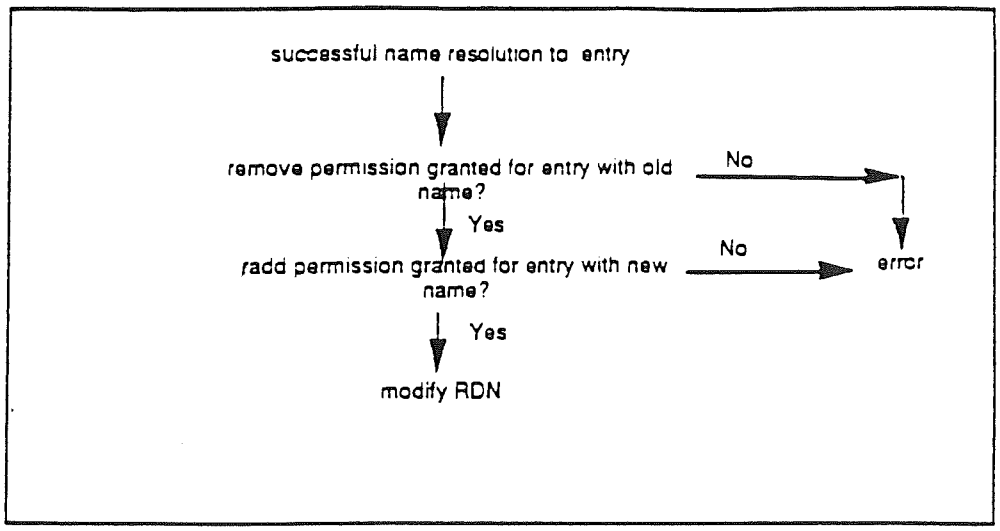


Fig 6.9. : L'opération modify entry.

Si l'opération échoue, une erreur est retournée :

- NameError en cas de problème noSuchObject.
- SecurityError en cas de problème insufficientAccessRights.

6.4.4. Modify RDN Operation.

Si le contrôle des accès de base est en application pour l'entrée en cours de renomination, le mécanisme suivant est d'application :

- si le but de l'opération est de modifier uniquement le dernier RDN du nom de l'entrée, alors la permission renommer, modifier, ou retirer est requise pour cette entrée. En plus, la permission ajouter est requise pour cette entrée fraichement renommée. Si une permission est absente, l'opération échoue.
- dans tous les autres cas, les permissions renommer et retirer sont requises pour l'entrée à renommer. De plus, les permissions ajouter et renommer sont requises pour l'entrée considérée dans sa nouvelle position au sein du DIT.

Le diagramme suivant illustre ce mécanisme.

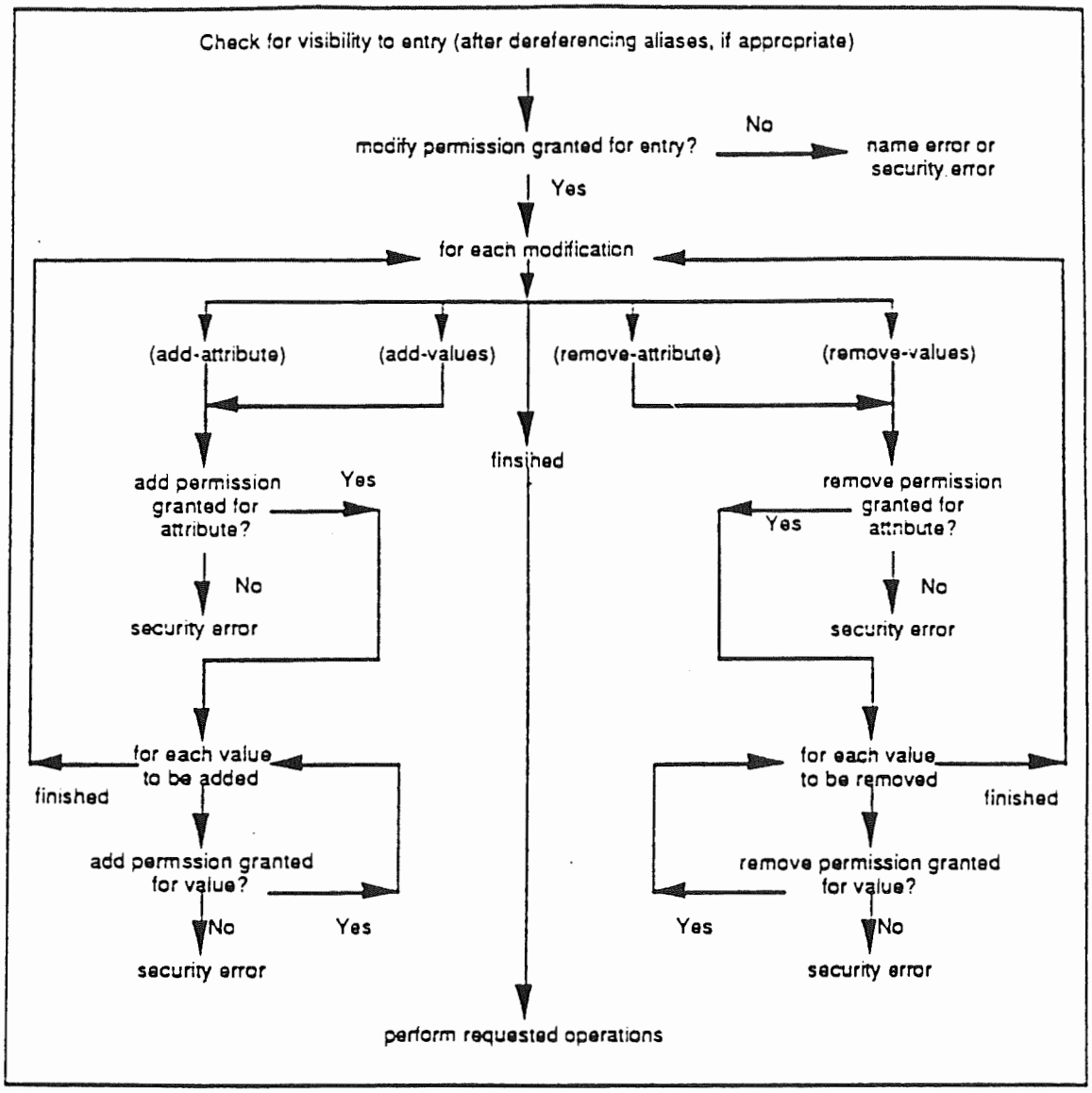


Fig 6.10. : L'opération modify RDN.

Si l'opération échoue, une erreur est retournée :

- NameError en cas de problème noSuchObject.
- SecurityError en cas de problème insufficientAccessRights.

7.1. Modèle d'Information des DSAs - DSA Information Model. X.518

7.1.1. Connaissances - Knowledge.

7.1.1.0. Introduction.

Le Directory est distribué en un nombre considérable de DSAs maîtres, chacun de ceux-ci contenant une partie du DIB pour laquelle il a une responsabilité administrative. En plus, ces DSAs et d'autres peuvent contenir des copies d'information, des copies de parties du DIB (cfr. chapitre 8).

Afin de supporter cette exigence opérationnelle, il est nécessaire que chaque DSA soit capable d'accéder à l'information contenue par le DIB associée à un nom. Si le DSA ne contient pas l'entrée ou la copie de l'entrée requise, il doit être capable d'interagir avec le DSA qui la contient, et cela soit directement, soit indirectement.

Quand un utilisateur du Directory indique que les copies d'information ne peuvent être utilisées pour satisfaire sa requête, le DSA en service doit pouvoir atteindre directement ou indirectement le DSA maître de cette information, càd le DSA contenant l'entrée associée à l'information requise.

Cette section sur les connaissances du Directory définit l'information opérationnelle nécessaire à un DSA (connaissance) pour mener à bien une requête de l'utilisateur.

7.1.1.1. Références de Connaissance - Knowledge References.

La connaissance est cette information opérationnelle, stockée par un DSA, qui représente une description partielle de la distribution des entrées et des copies d'entrée contenues dans d'autres DSAs. La connaissance est utilisée par un DSA pour déterminer le DSA approprié à contacter quand une requête reçue d'un DUA ou d'un autre DSA ne peut être satisfaite grâce à l'information stockée localement.

La connaissance consiste en différentes références de connaissance. Un référence de connaissance associe soit directement, soit indirectement, le nom d'une entrée du Directory avec un DSA contenant cette entrée ou une copie de celle-ci.

7.1.1.1.1. Catégories de Connaissances - Knowledge Categories.

Il existe deux catégories de références de connaissance :

- les références de connaissance "maître".
- les références de connaissance "shadow".

Les connaissances maîtres sont des renseignements sur le point d'accès au DSA maître pour un contexte de dénomination donné.

Les connaissances shadow sont des renseignements sur les DSAs contenant de l'information répliquée. Ces connaissances peuvent être distribuées par des DSAs fournisseurs aux DSAs consommateurs au moyen de procédures de réplication. Les connaissances shadow contiennent de l'information sur les points d'accès à un ensemble de DSAs shadow pour un contexte de dénomination.

Un DSA donné peut stocker des connaissances maîtres et shadow en même temps.

7.1.1.1.2. Types de Référence de Connaissance.

Un DSA peut contenir les types de références de connaissance suivants :

- référence supérieure.
- référence supérieure immédiate.
- référence subordonnée.
- référence subordonnée non-spécifique.
- référence croisée.

Une référence de connaissance d'un type particulier peut être une référence maître ou une référence shadow.

En plus, un DSA qui participe à un shadowing, soit comme fournisseur, soit comme consommateur, peut contenir un ou plusieurs types de référence de connaissance suivants :

- référence fournisseur.
- référence consommateur.

Les paragraphes suivants détaillent ces diverses références lorsqu'elles sont nouvelles ou si elles ont été modifiées depuis la recommandation de 1988.

7.1.1.1.2.1. Références Supérieures Immédiates.

Une référence supérieure immédiate consiste en :

- un préfixe de contexte de dénomination qui est immédiatement supérieur à celui contenu (comme entrées ou copies d'entrée) par le DSA stockant cette référence.
- un point d'accès au DSA contenant ce contexte de dénomination.

7.1.1.1.2.2. Références Subordonnées.

Une référence subordonnée consiste en :

- un préfixe correspondant à un contexte de dénomination immédiatement subordonné à celui contenu par le DSA stockant cette référence.
- un point d'accès au DSA contenant ce contexte de dénomination.

7.1.1.1.2.3. Références Fournisseur.

Une référence fournisseur contenue par un DSA consommateur (de shadowing) consiste en :

- un préfixe de contexte de dénomination copié par le DSA consommateur (de shadowing).
- un entier identifiant l'accord de shadowing que le DSA consommateur a établi avec le DSA fournissseur.
- un point d'accès au DSA fournisseur.

7.1.1.1.2.4. Références Consommateur.

Une référence consommateur contenue par un DSA fournisseur (de shadowing) consiste en :

- un préfixe de contexte de dénomination fourni par le DSA fournisseur.
- un entier identifiant l'accord de shadowing que le DSA fournisseur a établi avec le DSA consommateur.
- un point d'accès au DSA consommateur.

7.1.1.2. Connaissance Minimale.

Certaines requêtes ne peuvent être satisfaites qu'en utilisant des entrées originales (càd pas une copie). Cependant, quand l'utilisateur permet l'emploi de copies d'information, le Directory atteint un niveau de performance et de disponibilité supérieur.

Afin de réaliser ces deux approches, le Directory doit maintenir une quantité minimum de connaissances qui dépend de la configuration choisie.

L'objectif de ces exigences minimales est de permettre au processus de résolution des noms, dans le cas distribué, d'établir un chemin de références, comme une séquence continue de références de connaissance maîtres, vers tous les contextes de dénomination du Directory.

Au delà de ces exigences minimales, des connaissances additionnelles peuvent être employées pour établir d'autres chemins de références vers des copies de contextes de dénomination.

Voyons maintenant quelles sont ces connaissances minimales.

7.1.1.2.1. Connaissance Supérieure.

Chacun des DSAs n'étant pas un DSA de premier niveau doit maintenir une référence supérieure.

7.1.1.2.2. Connaissance Subordonnée.

Un DSA qui est maître d'un contexte de dénomination maintiendra des références subordonnées ou subordonnées non-spécifiques pour chacun des DSAs contenant (en maître) un contexte de dénomination immédiatement subordonné.

7.1.1.2.3. Connaissance Fournisseur.

Chaque DSA consommateur de shadowing maintiendra une référence fournisseur pour chaque DSA fournisseur de shadowing qui lui fournit une copie d'un contexte de dénomination. Si les connaissances subordonnées du DSA consommateur pour une copie de contexte de dénomination sont incomplètes, il utilisera sa référence fournisseur et/ou ses connaissances sur le DSA maître de contexte de dénomination afin d'établir un chemin de référence vers l'information subordonnée.

7.1.1.2.4. Connaissance Consommateur.

Un DSA fournisseur de shadowing maintiendra une référence consommateur pour chaque DSA consommateur de shadowing qu'il fournit d'une copie de contexte de dénomination.

7.1.1.3. DSAs de Premier Niveau - First Level DSAs.

Les exigences de connaissances minimales imposées à tous les DSAs présument que chaque DSA maintiendra au moins une référence supérieure. Le DSA référencié par une référence supérieure assume l'établissement du chemin de référence vers tous les éléments du DIT qui sont inconnus du DSA en question. Un DSA référencié par d'autres DSAs peut lui aussi maintenir une référence supérieure. Ce procédé de référence récursif s'arrête à l'ensemble des DSAs de premier niveau.

Un DSA de premier niveau est caractérisé par les éléments suivants :

- il ne possède pas de référence supérieure.
- il peut contenir un ou plusieurs contextes de dénomination immédiatement subordonnés à la racine du DIT.
- il maintient une référence subordonnée pour chaque contexte de dénomination immédiatement subordonné à la racine du DIT qu'il ne contient pas lui-même.

Les autorités administratives pour les DSAs de premier niveau sont communément responsables de l'administration des subordonnés immédiats de la racine du DIT. Les procédures gouvernant cette administration commune sont déterminées par des accords bilatéraux qui sont hors de la portée de ces recommandations.

7.1.2. Eléments de Base du Modèle d'Information des DSAs.

7.1.2.0. Introduction.

Le modèle d'information des DSAs décrit comment le Directory représente l'information au sujet des objets ayant un nom distingué et optionnellement des noms de type alias.

D'autre part, le modèle d'information des DSAs est spécialement concerné par les DSAs et l'information qui doit être contenue par ces mêmes DSAs afin que l'ensemble des DSAs, représentant le Directory, puissent réaliser ensemble ce modèle d'information.

Ceci concerne plus particulièrement :

- comment l'information du Directory (entrées de type objet et alias, sous-entrées) est située au sein des DSAs.
- comment les copies d'information du Directory peuvent être maintenues par les DSAs.
- quelle est l'information opérationnelle requise par les DSAs pour effectuer la résolution des noms.
- quelle est l'information opérationnelle requise par les DSAs pour s'engager dans le shadowing et utiliser l'information répliquée.

7.1.2.1. Entrées Spécifiques des DSAs et leurs Noms.

Dans le modèle d'information des DSAs, les "entrepots" d'information contenant l'information associée à un nom particulier portent le nom d'entrées spécifiques des DSAs (DSA Specific entries, DSEs en abrégé). Les entrées du Directory existent dans le modèle d'information des DSAs seulement comme éléments d'information à partir desquels peuvent être formés les DSEs. Les attributs opérationnels spécifiques au modèle d'information des DSAs composent l'autre variété d'information pouvant servir à la composition des DSEs.

Si un DSA contient de l'information concernant directement un nom, on dit qu'il connaît ou qu'il a connaissance de ce nom.

Pour chaque nom connu par un DSA, toute l'information contenue par ce DSA, directement associée à ce nom et autre que le nom lui-même, est représentée par un seul DSE.

L'ensemble des noms connus par un DSA auquel on ajoute l'information associée à chacun de ces noms est appelé arbre de l'information des DSAs (DSA Information Tree). La figure suivante illustre un exemple d'arbre de l'information des DSAs.

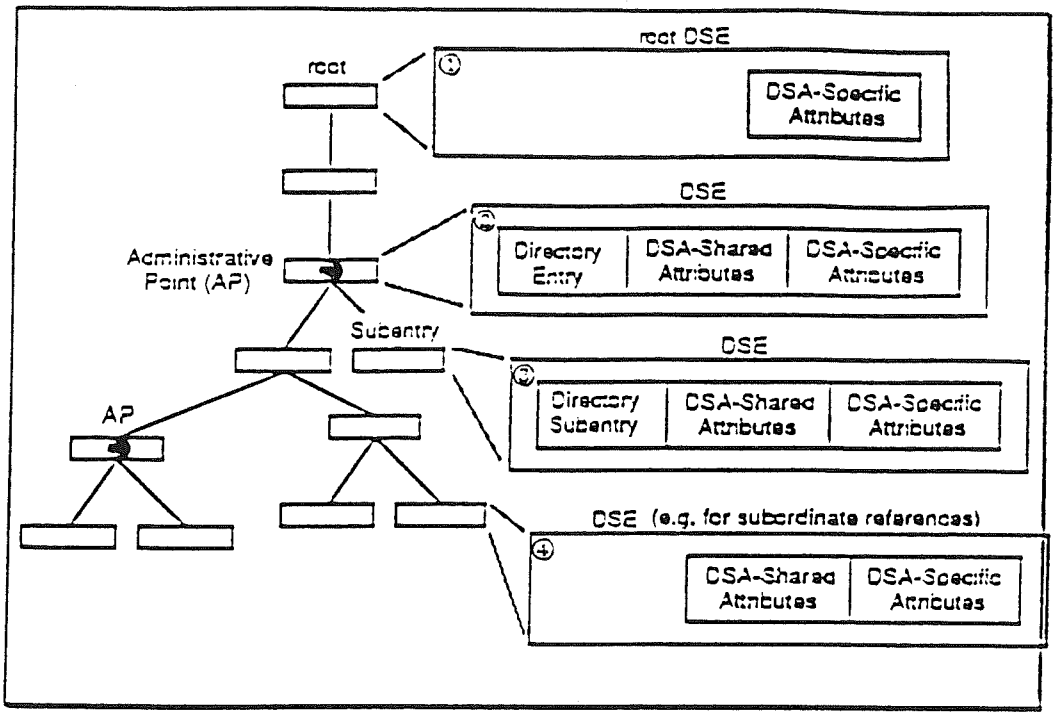


Fig 7.1.: Arbre de l'information des DSAs.

L'information minimale qu'un DSA peut associer à un nom, et donc connaître ce nom, consiste en une expression des buts pour lesquels ce nom est connu (i.e. le rôle joué par ce nom dans l'opération du DSA le connaissant). Ce but est représenté dans le modèle d'information des DSAs par un attribut spécifique au DSA (DSA-specific attribute) : le type de DSE.

En plus, un DSE peut contenir d'autres informations associées au nom, telle qu'une entrée ou une copie d'entrée, des attributs spécifiques aux DSAs ou des attributs partagés par les DSAs (DSA-shared attributes).

Un DSE peut aussi représenter directement une entrée du Directory, une partie d'une entrée ou pas d'information du tout. L'information contenue dans un DSE varie en fonction de son type ou de son but. En général, les différents types de DSEs qui peuvent apparaître dans un DSA sont les suivants (en relation avec la figure précédente) :

- un DSE représentant directement une entrée du Directory contient les attributs utilisateurs et opérationnels correspondant à l'entrée du Directory (comme représenté par le DSE 2). Le DSE peut aussi contenir des attributs spécifiques ou partagés par les DSAs.
- un DSE représentant une partie d'une entrée du Directory (comme résultat d'un shadowing) contient une partie de l'information utilisateur et opérationnelle, sous forme d'attributs, correspondant à cette entrée du Directory. Il contient aussi des attributs spécifiques aux DSAs et il peut contenir des attributs partagés par les DSAs.
- un DSE représentant seulement de l'information opérationnelle contient les attributs utilisateurs et opérationnels d'une sous-entrée du Directory (comme représenté par le DSE 3). Le DSE peut aussi contenir des attributs spécifiques ou partagés par les DSAS.
- un DSE représentant aucune information du Directory contient seulement des attributs partagés et/ou spécifiques aux DSAs (comme représenté par les DSE 1 et 4). Par exemple, un DSE représentant une référence subordonnée peut posséder un attribut partagé par les DSAs, qui indique le point d'accès au DSA maître, et un attribut spécifique aux DSAs indiquant que le DSE est une référence subordonnée.

7.1.2.2. Eléments de Base.

Un DSE comprend trois éléments de base : le type du DSE, un certain nombre d'attributs opérationnels de DSA et optionnellement, une entrée ou une copie d'entrée.

7.1.2.2.1. Attributs Opérationnels de DSA.

Deux variétés d'attribut opérationnel apparaissent dans le modèle d'information des DSAs et ne correspondent pas à de l'information contenue dans une entrée du Directory : les attributs partagés par les DSAs et les attributs spécifiques aux DSAs.

Un attribut partagé par les DSAs (DSA-shared attribute) est un attribut opérationnel dans le modèle d'information des DSAs associé à un nom particulier dont la ou les valeurs, si elles sont tenues par plusieurs DSAs, sont identiques. Un DSA peut maintenir une copie (shadow-copy) d'un attribut de ce type.

Un attribut spécifique aux DSAs (DSA-specific attribute) est un attribut opérationnel dans le modèle d'information des DSAs associé à un nom particulier dont la ou les valeurs, si elles sont tenues par plusieurs DSAs, ne doivent pas être nécessairement identiques. Un attribut de ce type représente de l'information opérationnelle qui est spécifique au fonctionnement du DSA le contenant. Un DSA ne peut pas tenir une copie (shadow-copy) d'un attribut spécifique aux DSAs.

7.1.2.2.2. Types de DSE - DSE Types.

Le type d'un DSE, représenté dans le modèle d'information des DSAs par un attribut opérationnel spécifique aux DSAs, indique le but particulier ou rôle joué par un DSE. Ce but est indiqué par un ensemble de bits (un ou plusieurs). Un DSE peut avoir plusieurs rôles. Ces bits ont un nom comme nous le verrons dans la suite de ce paragraphe.

Quand on dit qu'un DSE est de type x, cela signifie que le bit de nom x a été positionné (à 1).

Voyons maintenant ces différents bits ou différentes valeurs d'un type de DSE :

- root : le DSE racine contient des attributs spécifiques aux DSAs qui caractérisent le DSA. Le nom correspondant au DSE racine est un nom dégénéré consistant en une séquence de zéro RDN.
- glue : un DSE de type glue représente la connaissance d'un nom uniquement. Un DSA contenant un DSE de type préfixe de contexte ou de type référence croisée peut contenir des DSEs de type glue pour représenter le noms des supérieurs au DSE de type préfixe de contexte ou de type référence croisée, si aucune autre information opérationnelle n'est associée à ces noms.
- cp : ce type de DSE représente le préfixe de contexte (d'un contexte de dénomination).
- entry : ce DSE contient une entrée du Directory.
- alias : ce DSE contient une entrée de type alias.
- subr : ce DSE contient un attribut représentant une référence subordonnée.
- nssr : ce DSE contient un attribut représentant une référence subordonnée non-spécifique.
- supr : ce DSE contient un attribut représentant une référence supérieure à un DSA.
- xr : ce DSE contient un attribut représentant une référence croisée.

- admPoint : ce DSE correspond à un point administratif.
- subentry : ce DSE contient une sous-entrée.
- shadow : ce DSE contient une copie (shadow-copy) d'une entrée (ou d'une partie de celle-ci) ou une autre information (par exemple une connaissance) reçue d'un DSA fournisseur de shadow.
- immSupr : ce DSE contient un attribut représentant une référence supérieure immédiate.

L'utilisation de ceci, afin de représenter les aspects du modèle d'information des DSAs, est décrite à la section suivante.

7.1.3. Représentation de l'Information des DSAs.

Cette section traite de la représentation de l'information des DSAs. Elle décrit aussi la représentation de l'information opérationnelle des DSAs (connaissance), de l'information utilisateur et opérationnelle.

7.1.3.1. Représentation de l'Information Opérationnelle et Utilisateur.

Ce paragraphe spécifie la représentation de l'information utilisateur et opérationnelle du Directory dans le modèle d'information des DSAs.

7.1.3.1.1. Entrées de Type Objet - Object Entries.

Une entrée de type objet est représentée par un DSE de type entrée qui contient les attributs utilisateur et opérationnels associés à cette entrée du Directory. Le nom du DSE est le nom de l'entrée de type objet (i.e. son nom distingué).

Si le DSE contient une copy de l'entrée, le type du DSE est entrée et shadow.

7.1.3.1.2. Entrées de Type Alias - Alias Entries.

Une entrée de type alias est représentée par un DSE de type alias qui contient les attributs associés à l'entrée de type alias (i.e. les attributs RDN et l'attribut du nom de l'objet référencié par l'alias. Le nom du DSE est le nom de l'entrée de type alias.

Si le DSE contient une copie de l'entrée de type alias, le type du DSE est alias et shadow.

7.1.3.1.3. Point Administratif.

Un point administratif est représenté par un DSE de type entrée et admPoint (ou admPoint, entrée et cp) qui contient les attributs associés au point administratif. Le nom du DSE est celui du point administratif.

Si le DSE contient une copie de l'information d'un point administratif, le type du DSE est admPoint, entrée et shadow (ou admPoint, entrée, cp et shadow).

Dans le cas où une étendue administrative s'étend sur deux ou plusieurs DSAs, le point administratif dans les DSAs subordonnés peut ne pas être associé avec une entrée. Le DSE pour un point administratif dans un tel DSA sera du type admPoint et glue. Si le DSE représente un référence de connaissance (par exemple immSupr), le bit glue n'est pas positionné.

7.1.3.1.4. Sous-Entrées - Subentries.

Une sous-entrée est représentée par un DSE de type sous-entrée qui contient les informations opérationnelles associées à cette sous-entrée. Le nom du DSE est le nom de la sous-entrée.

Si le DSE contient une copie de la sous-entrée, le type du DSE est sous-entrée et shadow.

7.1.3.2. Représentation des Références de Connaissance.

Une référence de connaissance consiste en un DSE de type particulier qui contient un attribut opérationnel de DSA approprié correspondant et qui est identifié par un nom représentant une relation définie sur le contexte de dénomination maintenu par le DSA en question.

7.1.3.2.1. Types d'Attribut de Connaissance.

Les atrributs opérationnels des DSAs sont définis dans le modèle d'information des DSAs et expriment les idées suivantes pour un DSA :

- connaissance de son propre point d'accès.
- connaissance supérieure.
- connaissance spécifique (ses références subordonnées).
- connaissance non-spécifique (ses références subordonnées non spécifiques).

- connaissance de ses fournisseurs s'il est un consommateur de shadow.
- connaissance de ses consommateurs s'il est fournisseur de shadow.
- connaissance de shadow secondaire s'il est fournisseur de shadow.

7.1.3.2.1.1. Mon Point d'Accès - My Access Point.

Le type d'attribut opérationnel myAccessPoint est utilisé par un DSA pour représenter son propre point d'accès. Tous les DSAs maintiendront cet attribut dans leur DSE racine. Celui-ci n'a qu'une seule valeur et est administré par le DSA lui-même.

7.1.3.2.1.2. Connaissances Supérieures.

Le type d'attribut opérationnel connaissance supérieure (superiorKnowledge) est utilisé par un DSA n'étant pas de premier niveau pour représenter sa référence supérieure. Il s'agit d'un attribut spécifique aux DSAs. Tous les DSAs n'étant pas de premier niveau maintiendront cet attribut dans leur DSE racine. Celui-ci n'a qu'une seule valeur et est administré par le DSA lui-même.

7.1.3.2.1.3. Connaissances Spécifiques.

Une connaissance spécifique consiste en un point d'accès au DSA maître d'un contexte de dénomination et/ou un shadow DSA pour ce contexte de dénomination. Il est spécifique parce que le préfixe d'un contexte de dénomination est connu et est associé à l'information d'un point d'accès. Une connaissance spécifique est représentée par un attribut opérationnel de type connaissance spécifique (specificKnowledge). Celui-ci est un attribut partagé par les DSAs, n'a qu'une seule valeur et est administré par le DSA lui-même.

Le type d'attribut connaissance spécifique est contenu dans un DSE de type cp, subr, xr ou immSupr. Il est utilisé par un DSA contenant un contexte de dénomination (ou une copie de celui-ci) pour représenter le DSA maître (ou un DSA shadow pouvant effectuer ce rôle) de ce contexte. Il est aussi utilisé pour représenter des références subordonnées, croisées ou supérieures immédiates.

7.1.3.2.1.4. Connaissances Non-Spécifiques.

Une connaissance non-spécifique consiste en points d'accès à un DSA maître pour un ou plusieurs contextes de dénomination et/ou shadow DSAs pour le ou les mêmes contextes de dénomination. Elle est non-spécifique parce que le(s) préfixe(s) n'est(ne sont) pas connu(s). Le supérieur immédiat au contexte(s) de dénomination est connu et l'information de point d'accès est associée à son nom. Une connaissance non-spécifique est représentée par le type d'attribut opérationnel connaissance non-spécifique (nonSpecific- Knowledge). Il s'agit d'un attribut partagé par les DSAs à valeurs multiples et administré par le DSA lui-même.

L'attribut de type connaissance maître non-spécifique (nonSpecificMasterKnowledge) est contenu dans un DSE de type nssr. Il est utilisé pour représenter des références subordonnées non-spécifiques.

7.1.3.2.1.5. Connaissances Fournisseur.

Une connaissance fournisseur d'un DSA consommateur de shadowing consiste en point(s) d'accès et identifiant(s) d'accord de shadowing pour son(ses) fournisseur(s) de copie(s) de contexte de dénomination. Cette connaissance fournisseur est représentée par un attribut de type connaissance fournisseur (supplierKnowledge). Il s'agit d'un attribut spécifique aux DSAs, multivalué et administré par le DSA lui-même.

L'attribut de type connaissance fournisseur est contenu dans un DSE de type cp. Il est utilisé pour représenter une ou plusieurs références fournisseurs. Tous les DSAs consommateurs de shadowing doivent maintenir une valeur de cet attribut pour chacun des accords de shadowing dans lequel ils s'engagent comme consommateurs.

7.1.3.2.1.6. Connaissances Consommateur.

Une connaissance consommateur d'un DSA fournisseur de shadowing consiste en point(s) d'accès et identifiant(s) d'accord de shadowing pour le(s) consommateur(s) de copie(s) de contexte de dénomination qui leur est(sont) procuré(s) par ce fournisseur.

Une connaissance consommateur est représentée par un attribut opérationnel de type connaissance(consumerKnowledge). Il s'agit d'un attribut spécifique aux DSAs, à valeurs multiples et administré par le DSA lui-même.

L'attribut de connaissance consommateur est contenu dans un DSE de type cp. Il est utilisé pour représenter une ou plusieurs références consommateurs. Tous les DSAs fournisseurs de shadowing doivent maintenir une valeur de cet attribut pour chaque accord de shadowing dans lequel ils s'engagent comme fournisseurs.

7.1.3.2.1.7. Connaissances de Shadowing Secondaire.

Une connaissance de shadowing secondaire consiste en une information qu'un DSA fournisseur de shadowing peut choisir de maintenir au sujet de DSAs consommateurs qui sont engagés dans une procédure de shadowing secondaire avec lui. Une connaissance de shadowing secondaire est représentée par un attribut opérationnel de type shadowing secondaire (secondaryShadows). Il s'agit d'un attribut spécifique aux DSAs, multivalué et administré par le DSA lui-même.

Un DSA fournisseur de shadowing peut obtenir de l'information lui permettant de fabriquer les valeurs de cet attribut en s'engageant dans un accord additionnel de shadowing avec chacun de ses DSAs consommateurs (accord de shadowing "renversé"). Cet accord renversé entraînera l'administration par le consommateur de deux attributs opérationnels, une connaissance consommateur et une connaissance de shadowing secondaire, associés au préfixe de contexte de dénomination copiés (shadow-copied) par le consommateur. Après réception de mises-à-jour de ces deux attributs, le DSA fournisseur mettra à son tour à jour son propre attribut de shadowing secondaire, associé à ce préfixe de contexte afin d'inclure toutes les valeurs de shadowing secondaire du consommateur et les valeurs additionnelles créées par le composant point d'accès de l'attribut connaissance consommateur. L'utilisation récursive de cette procédure permet à un DSA maître d'un contexte de dénomination de tout connaître au sujet de ses DSAs consommateurs de shadowing secondaire.

L'implémentation et l'utilisation du shadowing secondaire est optionnelle.

7.1.3.2.2. Types de Référence de Connaissance.

Ce paragraphe spécifie la représentation des connaissances dans le modèle d'information des DSAs.

7.1.3.2.2.1. Self Référence.

Une self référence représente la connaissance d'un DSA au sujet de son propre point d'accès. Elle consiste en une valeur de l'attribut mon point d'accès (myAccesPoint) contenu dans le DSE racine du DSA, le DSE de type racine.

7.1.3.2.2.2. Référence Supérieure.

Une référence supérieure consiste en un DSE de type supr et racine (root) qui contient un attribut de type connaissance supérieure.

7.1.3.2.2.3. Référence Supérieure Immédiate.

Une référence supérieure immédiate consiste en un DSE de type immSupr qui contient un attribut de connaissance spécifique. Le nom du DSE contenant cet attribut correspond au préfixe du contexte de dénomination administré par le DSA référencié.

Si le DSE contenant cette référence supérieure immédiate est une copie d'information, reçue d'un fournisseur de shadowing, le DSE est de type immSupr et shadow.

7.1.3.2.2.4. Référence Subordonnée.

Une référence subordonnée consiste en un DSE de type subr qui contient un attribut de type connaissance spécifique (specific Knowledge). Le nom du DSE contenant cet attribut correspond au préfixe de contexte de dénomination maintenu par le DSA subordonné référencié.

Si le DSE contenant la référence subordonnée est une information copiée, reçue d'un DSA fournisseur de shadowing, le type de ce DSE est subr et shadow.

7.1.3.2.2.5. Référence Subordonnée Non-Spécifique.

Une référence subordonnée non-spécifique consiste en un DSE de type nssr et entrée qui contient un attribut de type connaissance non-spécifique (nonSpecificKnowledge). Le nom du DSE contenant cet attribut correspond au nom formé par élimination du dernier RDN du préfixe de contexte de dénomination maintenu par le DSA subordonné référencié.

Si le DSE contenant la référence subordonnée non-spécifique est une information copiée, reçue d'un DSA fournisseur de shadowing, le type de ce DSE est nssr, entrée et shadow.

7.1.3.2.2.6. Référence Croisée.

Une référence croisée consiste en un DSE de type xr qui contient un attribut de type connaissance spécifique. Le nom du DSE contenant cet attribut correspond au préfixe de contexte de dénomination maintenu par le DSA référencié.

Si le DSE contenant la référence croisée est une copie d'information, reçue d'un DSA fournisseur de shadowing, le type de ce DSE est xr et shadow.

7.1.3.2.2.7. Référence Fournisseur.

Une référence fournisseur consiste en un DSE de type cp qui contient un attribut de type connaissance fournisseur. Le nom du DSE contenant cet attribut correspond au préfixe de contexte de dénomination copié.

7.1.3.2.2.8. Référence Consommateur.

Une référence consommateur consiste en un DSE de type cp qui contient un attribut de type connaissance consommateur. Le nom du DSE contenant cet attribut correspond au préfixe de contexte de dénomination copié.

7.1.3.3. Représentation des Noms et Contextes de Dénomination.

7.1.3.3.1. Noms et DSEs de Type Glue.

Comme décrit au point 7.1.1.2., l'information minimale qu'un DSA peut associer à un nom est, en fait, la raison pour laquelle il connaît, contient ce nom, représenté par un DSE contenant une valeur pour l'attribut dseType. Quand un DSE ne contient que cette information minimale, la valeur de l'attribut dseType sera glue. Dans ce cas, le DSE ne maintiendra pas une entrée ou une sous-entrée (copie d'entrée ou de sous-entrée) ou un attribut de connaissance.

Les DSEs de type glue apparaissent dans le modèle d'information des DSAs pour représenter des noms connus par un DSA comme conséquence du maintient d'autres informations associées à d'autres noms.

En plus de tout ceci, il existe un type de DSE glue composite dans le modèle d'information des DSAs. Le type de DSE glue et admpoint est employé dans un DSA qui contient une copie (shadow copy) d'une ou plusieurs sous-entrées dans le cas ou il ne contiendrait pas de copie de l'entrée du point administratif. Ce DSE représente la connaissance du fait que le supérieur immédiat de la sous-entrée est un point administratif.

7.1.3.3.2. Contextes de Dénomination.

Un contexte de dénomination consiste en un préfixe (de contexte), en un sous-arbre de zéro ou plusieurs entrées subordonnées au préfixe (la racine du sous-arbre), et, s'il existe des contextes de dénomination subordonnés à celui-ci, en des références subordonnées spécifiques ou non suffisantes pour constituer une connaissance subordonnée complète.

Le préfixe de contexte est représenté par un DSE de type cp et entrée (ou cp, entrée et admpoint si le préfixe correspond à un point administratif).

La copie d'un contexte de dénomination est représentée comme ci-dessus, à la seule exception que le bit shadow est positionné dans chaque DSE.

7.1.3.3.3. Exemple.

La figure suivante illustre un exemple de la translation d'une partie du DIT (qui correspond à un contexte de dénomination) dans le modèle d'information des DSAs. En plus de l'information sur le contexte de dénomination lui-même, le DSE racine du DSA contenant sa référence supérieure (ceci n'est pas vrai pour un DSA de premier niveau), un DSE glue et un DSE représentant une référence (soit une référence croisée, soit une référence supérieure immédiate) à un contexte de dénomination immédiatement supérieur sont repésentées sur ce schéma.

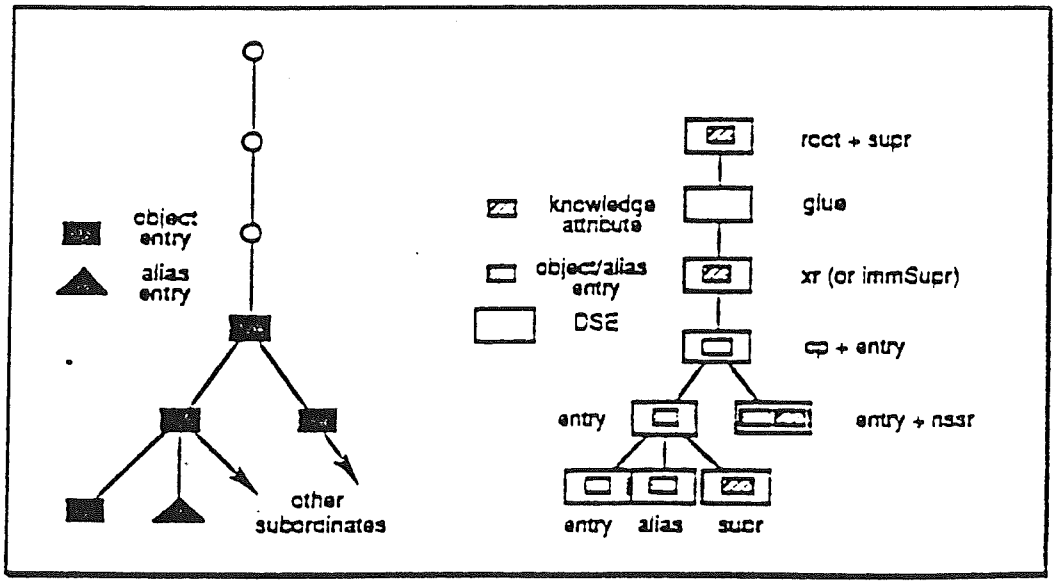


Fig 7.2. : DSEs pour un contexte de dénomination.

7.2. Modèle Opérationnel du Directory Distribué. [X.518]

7.2.1. Modèle d'Interaction des DSAs.

Comme nous l'avons vu lors de la présentataion des recommandations X.500 1988, trois modes d'interaction entre les DSAs ont été définis. La nouvelle recommandation nous fait part de quelques modifications à ces trois modes.

Les trois modes sont : le chaînage unique, le chaînage multiple et le "referral". Le chaînage est utilisé quand un DSA essaye de satisfaire une requête demandée par un DUA ou un autre DSA ; le "referral" est utilisé dans le cas où le DSA retourne de l'information de connaissance au demandeur.

7.2.1.1. Le Chaînage Unique - Uni-Chaining.

Le chaînage est utilisé par un DSA pour passer la requête à un autre DSA quand le DSA initiateur a des connaissances sur le contexte de dénomination tenu par le DSA récepteur ou une référence fournisseur ou maître.

7.2.1.2. Le Chaînage multiple - Multi-Chaining.

Ce mode d'interaction est utilisé par un DSA pour transferrer plusieurs requêtes en cours qui sont le résultat d'un requête initiale, comme résultat d'une décomposition de requête ou de multi-casting.

7.2.1.2.1. Le chaînage multiple parallèle.

Avec le multi-chaînage parallèle, le DSA transferre plusieurs requêtes en cours simultanément. Cette pratique peut augmenter la performance et en présence de shadowing, elle peut conduire à la production de multiples résultats.

7.2.1.2.2 Le chaînage multiple séquentiel.

Avec le multi-chaînage séquentiel, le DSA transferre une requête en cours à la fois, et attend le résultat de cette requête avant d'envoyer la suivante. Cette méthode n'est pas le mode d'interaction le plus rapide.

7.2.1.2.3. Le multi-casting.

Le multi-casting est une forme de chaînage multiple. Les références subordonnées non-spécifiques ne maintiennent pas le RDN du contexte de dénomination subordonné référencié. Dès lors, le DSA référençiant est incapable de dire quel DSA subordonné maintient tel ou tel contexte de dénomination. Durant la phase de résolution des noms, un DSA rencontrant une référence subordonnée non-spécifique doit d'abord envoyer une requête identique à chaque DSA qui lui est subordonné. Au pire, un DSA et un seul sera capable de continuer la résolution des noms, les autres renverront une erreur de type unableToProceed. Le procédé envoyant plusieurs requêtes identiques aux DSAs subordonnés lors de la rencontre d'une référence subordonnée non-spécifique, pendant la résolution des noms est appelé multi-casting.

7.2.1.2.4. Décomposition de requête.

La décomposition de requête, qui est une autre forme de chaînage multiple, est un procédé effectué de manière interne par un DSA, avant toute communication avec un autre DSA. Une requête est décomposée en plusieurs sous-requêtes différentes accomplissant chacune une partie de la tâche originelle. Un tel procédé ne peut être employé que pendant l'évaluation d'une opération list ou search. Après la décomposition, chacune des sous-requêtes est envoyée suivant un mode de chaînage aux autres DSAs afin de continuer la tâche, ou un résultat partiel peut être retourné au demandeur.

7.3. Procédures Distribuées. [X.518]

7.3.1. Comportement du Directory Distribué.

7.3.1.1 Phases de la Procédure d'une Opération.

Dans le cas des opérations read, compare, list , search, modify entry et remove entry, la résolution des noms s'effectue sur le nom fourni par les arguments de l'opération. Dans le cas des opérations add entry et modify RDN, la résolution des noms s'effectue sur le nom de l'entrée immédiatement supérieure à celle fournie par les arguments de l'opération.

En se basant sur son arbre d'information local et sur l'information de connaissance contenue en son sein, un DSA est capable de décider si la résolution des noms peut être poursuivie par un autre DSA ou bien si le nom est erroné.

Les opérations qui impliquent une simple interrogation d'entrée - read et compare - peuvent être effectuées entièrement au sein du DSA dans lequel l'entrée a été localisée. Le lecteur attentif remarquera que les autres opérations ne peuvent s'effectuer entièrement au sein d'un DSA à cause de l'emploi du shadowing.

Les opérations qui impliquent une interrogation multiple d'entrée - list et search - nécessitent la localisation de subordonnés de l'entrée cible, qui peuvent ou ne peuvent pas résider au sein du même DSA. S'ils ne se trouvent pas tous dans le même DSA, les opérations doivent être redirigées vers les DSAs spécifiés par les références subordonnées, subordonnées non-spécifiques , fournisseurs ou maîtres afin de compléter le procédé d'évaluation.

7.3.1.2. Gestion du cyclage - loop handling.

Le DIT peut se trouver dans un état causant un cyclage. Une cause de ce cyclage est l'utilisation inappropriée des alias, de telle façon que l'aliasedObjectName dans une suite d'aliases forme une boucle. Une autre cause de cyclage potentiel provient de la mauvaise configuration des références de connaissance.

Dans le contexte d'une opération particulière du Directory, un cyclage apparaît si l'opération se retrouve dans un état déjà rencontré. l'état d'une opération est défini de la manière suivante :

- le nom du DSA traitant l'opération en cours.
- le nom de l'objet cible comme contenu dans les arguments de l'opération.
- l'operationProgress.

Ceci ne signifie pas que l'opération ne puisse pas "repasser" plusieurs fois par le même DSA. Cependant cela veut bien dire qu'un DSA n'effectuera pas la même opération dans le même état plusieurs fois.

Le cyclage est régulé en utilisant l'argument traceInformation défini lors de la recommandation de 1988, qui capture la séquence d'états d'une opération particulière qui ont déjà été effectués. Deux stratégies ont été définies pour parer aux cyclages :

- détection de cyclage.
- prévention de cyclage.

7.3.1.2.1. Détection de Cyclage.

Lors de la réception d'une opération du Directory, un DSA doit initialement valider cette opération afin de s'assurer qu'elle peut progresser. Une tâche importante de la validation est la recherche de cyclage, en déterminant si le stade courant de l'opération apparaît déjà dans la séquence d'états enregistrée dans l'argument de traceInformation pour cette opération. Cette étape de vérification de cyclage est appelée détection de cyclage.

7.3.1.2.2. Prévention de Cyclage.

La prévention de cyclage requiert qu'un DSA, immédiatement avant d'envoyer l'opération à un autre DSA, détermine si l'état qui surviendrait apparaît dans la séquence d'états enregistrée dans l'argument de traceInformation pour cette opération.

Cette stratégie préventive n'est pas toujours possible, (quand il n'y a pas d'argument de traceInformation, i.e. l'opération vient d'un DUA).

7.3.2. L'Opération Dispatcher.

L'opération dispatcher est la principale procédure de contrôle d'un DSA. Il guide chaque opération à travers les trois phases d'une requête. Dès lors, l'opération dispatcher utilise un ensemble de procédures afin d'effectuer la requête. Ces procédures sont représentées à la figure suivante :

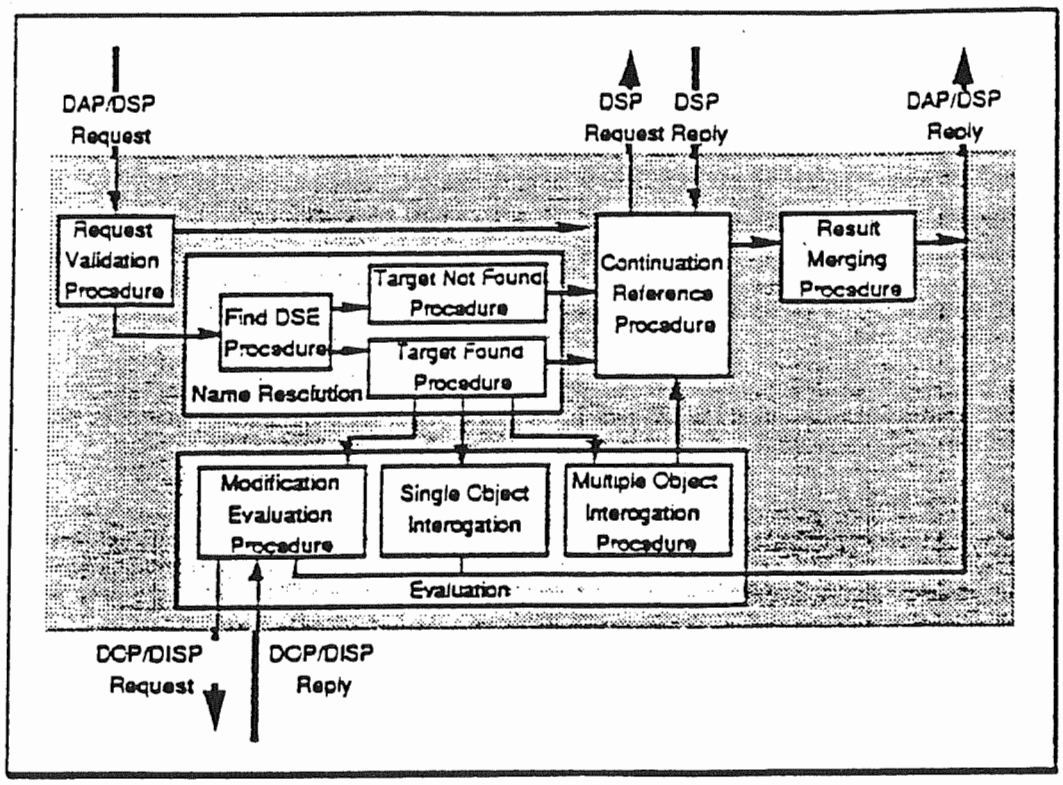


Fig 7.3.: L'opération dispatcher.

7.3.2.1. Concepts généraux.

7.3.2.1.1. Procédures.

Chaque procédure utilisée par l'opération dispatcher consiste en la définition de son interface en termes d'arguments, résultats et erreurs et en une description de la procédure elle-même.

7.3.2.1.2. Utilisation des Structures de Données Communes.

Toutes les procédures utilisent quelques structures de données qui sont valables durant le processus d'une opération au sein de l'opération dispatcher. Ces structures de données servent à coordonner le flux de données. La plupart de ces structures sont associées à l'argument de l'opération et au résultat devant être créé pour l'opération.

Les structures de données additionnelles suivantes sont définies au sein de l'opération dispatcher :

- continuationList : un ensemble de références créé par les procédures Target Not Found et Evaluation.
- administrativePoints : une liste de références aux DSEs, qui sont rassemblées durant la résolution des noms.

7.3.2.1.3. Intéractions externes.

L'opération dispatcher possède une et une seule sortie externe où les sous-requêtes DSP chaînées sont envoyées à un autre DSA sur ordre de la fonction de résolution des noms ou de décomposition des requêtes. Cet envoi prend lieu et place dans la procédure de référence de continuation (continuation reference). Les autres procédures créent une ou plusieurs continuationReferences au lieu de lancer des opérations chaînées. Ces références sont ajoutées à la continuationList.

Des interactions DOP ou DISP (Directory Information Shadow Protocol) peuvent résulter la procédure d'évaluation ou de modification d'évaluation.

7.3.2.1.4. Erreurs.

A chaque stade du processus, une erreur pourrait être détectée durant l'exécution de sous-procédure. L'erreur identifiée au sein de cette sous-procédure est normalement retournée au demandeur selon le protocol d'erreur correspondant.

Dans ce cas l'opération dispatcher est immédiatement terminée.

Alternativement, une procédure peut choisir de poursuivre malgré les erreurs. Dans ce cas, la procédure continue son exécution et aucune erreur n'est retournée au demandeur.

7.3.2.2. Procédure de l'Opération Dispatcher.

La procédure qui est exécutée par l'opération dispatcher pour chacune des requêtes est définie par les étapes suivantes :

- valider certains aspects des arguments de l'opération (procédure request validation).

- résoudre le nom de l'objet cible en exécutant la procédure Find DSE.

- exécuter, si le nom a été complètement résolu, la procédure Target Found qui ensuite appelle la procédure Modification Evaluation, Single Object Interrogation ou Multiple Object Interrogation, en accord avec l'opération à exécuter. Si l'entrée cible n'est pas utilisable pour lui appliquer l'opération (cas des shadow), une référence de continuation est créée et la procédure Continuation Reference est invoquée.

- exécuter, si le nom n'a pas été correctement résolu, la procédure Objet Not Found afin de créer un ensemble de références de continuation (continuationReferences) dans la liste de continuation (continuationList). Ensuite appliquer la procédure Continuation Reference.

- appeler la procédure d'évaluation d'objet multiple. Cette procédure crée une liste de références de continuation dans la continuationList qu'elle ajoute à l'ensemble des informations demandées pour chaque entrée. Ces références de continuation sont analysées par la procédure Continuation Reference.

- appeler la procédure Continuation Reference qui , comme dit précédemment, analyse les références de continuation de la continuationList et dès lors peut émettre une ou plusieurs sous-requêtes chaînées à d'autres DSAs et attendre leru réponse respective.

- exécuter la procédure Result Merging, si des sous-requêtes ont été émises. Cette procédure collecte les résultats de chacune des sous-requêtes afin de former un résultat unique, à l'opération, qui est donc retourné au demandeur.

7.3.2.3. Vues Générales des Procédures.

Cette section donne au lecteur un aperçu général sur le fonctionnement de base des diverses procédures évoquées ci-dessus.

7.3.2.3.1. Validation de la Requête - Request validation Procedure.

Cette procédure est appelée avant même que l'opération ne soit testée en ce qui concerne le cyclage et la sécurité. De plus, cette procédure fournit un ensemble de paramètres "défauts" pour les arguments de chaînage quand ceux-ci ne sont pas produits (dans le cas où l'opération provient directement d'un DUA).

7.3.2.3.2. Recherche du DSE - Find DSE Procedure.

Cette procédure cherche à faire correspondre les composants du nom de l'objet cible avec les DSEs maintenus localement afin de résoudre le nom de l'objet cible. Si un DSE de type alias est rencontré, l'alias est déréférencié (si cela est autorisé) et les procédures redémarrent avec un nouveau nom à résoudre.

La procédure fournit comme résultat :

- soit le DSE cible, qui est retourné, et la procédure cible trouvée (Target Found) est appelée.

- soit un DSE intermédiaire et la procédure cible non trouvée (Target not found) est appelée.

La procédure peut s'interrompre suite à bon nombre d'erreurs. Dans ce cas le protocol d'erreur associé est retourné au demandeur de l'opération et l'opération dispatcher se termine.

7.3.2.3.3. Cible Non Trouvée - Target Not Found Procedure.

Cette procédure évalue le DSE intermédiaire trouvé et crée un ensemble de références de continuation (continuationReferences) dans la liste de continuation (continuationList), basé sur les références de connaisance qui ont été détectées durant la procédure recherche DSE. Cette référence est envoyée comme argument à la procédure référence de continuation (Continuation Reference).

La procédure peut détecter un certain nombre d'erreurs, qui sont retournées suivant un protocol d'erreur adéquat au demandeur et l'opération dispatcher se termine.

7.3.2.3.4. Cible Trouvée - Target Found Procedure.

Cette procédure vérifie si le DSE trouvé est "acceptable" pour l'opération en question (par exemple dans le cas du shadowing). Ceci peut inclure une vérification pour tout le sous-arbre d'information répliquée situé en dessous de l'objet cible, dans le cas d'une opération à objets multiples (par exemple une recherche dans un sous-arbre).

Si l'entrée découverte est acceptable, la procédure appropriée d'évaluation de l'opération est appelée. Par contre, si elle n'est pas acceptable, une référence de continuation est créée, pointant vers le fournisseur (ou maître) de l'information, dans la liste de continuation et la procédure référence de continuation est invoquée.

7.3.2.3.5. Interrogation d'Objet Simple - Single Object Interrogation Procedure.

Cette procédure est appelée pour effectuer réellement les opérations n'affectant qu'une seule entrée, comme read et compare. Ceci est fait en exécutant une sous-procédure spécifique pour chacune de ces opérations. Après terminaison, une réponse (résultat ou erreur) est créée par les procédures et est retournée au DSA ou DUA demandeur.

7.3.2.3.6. Interrogation d'Objets Multiples - Multiple Object Interrogation Procedure.

Cette procédure est exécutée pour effectuer réellement les opérations qui affectent plusieurs entrées, qui peuvent ou ne peuvent pas se trouver au sein du même DSA. Ceci est fait en exécutant une sous-procédure spécifique définie pour les opérations list et search. Cette procédure crée un résultat local de l'opération d'évaluation et optionnellement un ensemble de références de continuation dans la liste de continuation. Si cette liste de continuation est vide à la fin de cette procédure, le résultat créé est directement retourné au DSA ou au DUA demandeur. Dans le cas contraire, ces références de continuation sont fournies à la procédure de référence de continuation.

7.3.2.3.7. Evaluation de Modification - Modification Evaluation Procedure.

Cette procédure est effectuée pour exécuter les opérations de modification comme AddEntry, RemoveEntry, ModifyEntry et ModifyRDN. Ceci est fait en exécutant une sous-procédure spécifique pour chacune de ces opérations. Pendant l'exécution de ces sous-procédures, des requêtes peuvent être émises vers d'autres DSAs. Après une fin heureuse, un résultat est retourné au DUA ou DSA demandeur.

7.3.2.3.8. Référence de Continuation - Continuation Reference Procedure.

Cette procédure analyse les références de continuation créées par d'autres procédures dans la liste de continuation et les résoud en émettant une sous-requête chaînée ou en créant une référence de continuation pour les parties non-explorées, ou en renvoyant une erreur (referral error) apparue lors de la résolution des noms au DSA ou DUA demandeur.

Quand tous les résultats ou erreurs pour toutes les sous-requêtes ont été reçus, la procédure de collecte des résultats est appelée.

7.3.2.3.9. Collecte des Résultats - Result Merging Procedure.

Cette procédure combine les résultats d'opérations locales aux résultats reçus de sous-requêtes chaînées. Si une sous-requête renvoie comme résultat une erreur, cette procédure détermine comment cette erreur doit être prise en compte. S'il reste des références de continuation dans la liste de continuation qui n'ont pas encore été analysées par la procédure de référence de continuation, celles-ci sont inclues dans le résultat de l'opération (comme remarque sur les parties non-explorées du DIT).

Le résultat aggrégé est retourné au DUA ou DSA demandeur.

7.3.3. Remarque.

Nous ne rentrerons pas dans le détail de fonctionnement de chacune de ces procédures car elles pourraient à elles seules faire l'objet d'un mémoire.

8. La Nouvelle Recommandation X.5rp.

8.1. Introduction.

Des copies d'information ou information répliquées peuvent exister au sein du Directory. Lors de l'exposé de cette nouvelle recommandation, nous parlerons de deux mécanismes de réplication : le caching et le shadowing. Remarquons directement que les procédures de caching sont considérées comme étant entièrement gouvernées par des politiques locales et dès lors hors de la portée de ce mémoire.

Le déploiement de copies additionnelles de l'information contenue dans les entrées du Directory peut être utilisé dans le but d'une amélioration du service fourni par le Directory. Ces améliorations sont de deux types :

- augmenter les performances des systèmes du Directory en plaçant l'information "plus près" de certains utilisateurs particuliers du Directory.
- augmenter la disponibilité des services du Directory en introduisant de l'information redondante au sein du Directory de telle manière qu'une panne d'un des composants de ce Directory ne supprime pas l'accès à toute une partie du DIT.

Le déploiement de copies additionnelles peut être utilisé dans les systèmes de gestion du Directory. Les améliorations à ce niveau sont de deux types :

- faciliter la distribution de l'information opérationnelle.
- fournir l'opportunité de restaurer l'état du Directory après un problème grave en reconstruisant l'information stockée dans un composant du Directory à partir de copies de cette information contenues dans un autre composant du Directory.

8.2. Le Caching.

Le caching est une des méthodes de réplication de l'information applicable au Directory. Comme nous l'avons fait remarquer dans l'introduction de cette section, les méthodes de caching ne sont pas exposées dans cette recommandation X.5rp car elles sont purement locales. Plus particulièrement, les procédures de création de copies "cachées" et le maintien de la consistance entre ces dites copies et leurs entrées correspondantes sont sujet à des choix locaux.

8.3. Le Shadowing.

Le shadowing est une autre méthode de réplication d'information au sein du Directory. Cependant, avant que le shadowing puisse prendre place, les administéchoueurs des deux DSAs en question doivent parvenir à un accord sur les termes selon lesquels le shadowing prendra place. Cette négociation est faite avant n'importe quel échange protocolaire. L'accord en lui même est décrit à la section 8.4.1. de ce chapitre.

Dans la littérature, la copie d'information selon ce procédé porte le nom de shadowed information. Dans la suite, nous parlerons d'information répliquée car nous n'avons pas trouvé d'équivalent français satisfaisant.

8.3.1. Modèle Fonctionnel du Shadowing.

L'information répliquée est stockée par différents DSAs dans le Directory. D'une manière standardisée, un DSA peut assurer le rôle de shadow supplier et est dès lors la source de l'information répliquée ou peut assurer le rôle de shadow consumer et est donc le récepteur ou consommateur de l'information répliquée. Remarquons immédiatement que lors d'activités de réplication standardisées, le rôle joué par un DSA est toujours en correspondance avec le rôle joué par un autre DSA, ce dernier rôle étant réciproque au premier.

Il existe deux types de shadowing. Un que nous appelerons primaire (primary shadowing) et l'autre secondaire (secondary shadowing). La politique du shadowing primaire force tous les DSAs consommateurs d'information répliquée à recevoir les mises-à-jour directement du DSA (master DSA) possesseur de l'information répliquée. Par contre, la politique du shadowing secondaire permet à un DSA consommateur d'information répliquée de jouer le rôle de fournisseur de cette information avec d'autre DSAs n'ayant pas un accord direct avec le DSA maître de l'information répliquée.

Voyons maintenant ces deux politiques plus en détail.

8.3.2. Le Shadowing Primaire - Primary Shadowing.

La politique de shadowing primaire requiert que chacun des consommateurs d'information répliquée reçoive ses mises-à-jour directement du fournisseur de l'information répliquée. Le DSA fournisseur de l'information répliquée est habituellement appelé le DSA maître (master DSA) de l'unité d'information répliquée. Chacun des DSAs consommateurs d'information répliquée possède un accord de shadowing (shadowing agreement discuté en 8.4.1.) avec le DSA maître de l'unité de réplication en question.

De cette façon, un DSA consommateur d'information répliquée n'a pas l'autorité nécessaire à la gestion de cette unité de réplication et dès lors, toutes les opérations de modification concernant cette unité d'information répliquée sont redirigées vers le DSA maître.

Il va sans dire que les seules opérations permises sur une unité d'information répliquée au sein d'un DSA consommateur, sont les opérations de lecture et de recherche. Le DSA maître est responsable de l'envoi de toutes les mises-à-jour concernant une unité de réplication au sein d'un DSA consommateur de cette même unité d'information répliquée.

La figure 8.1. illustre une politique de shadowing primaire.

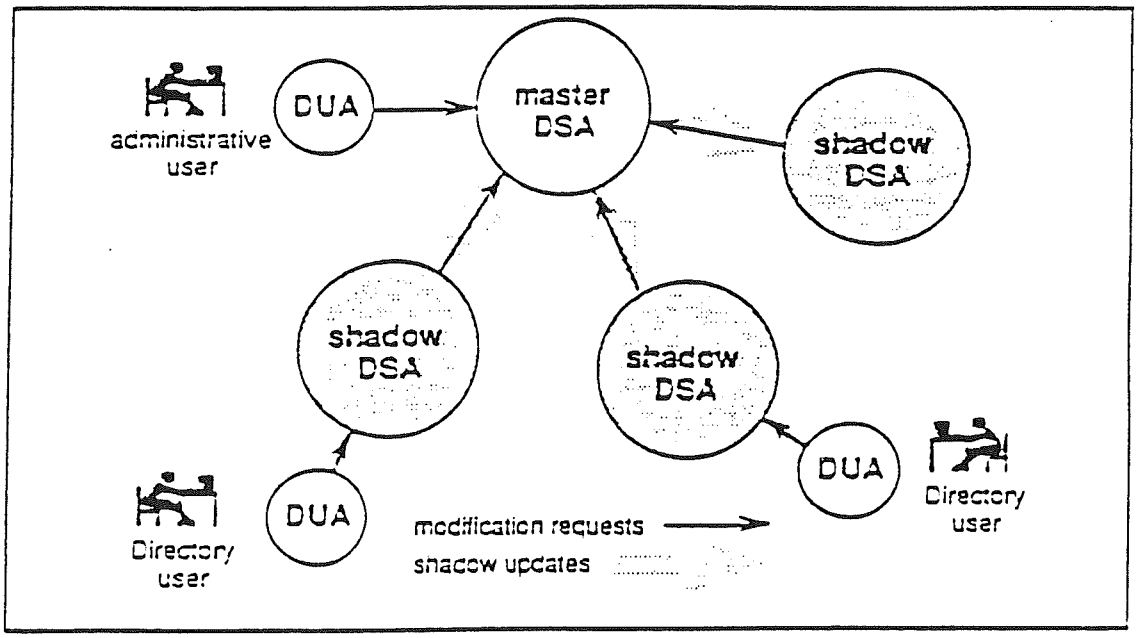


Fig 8.1. : Politique de shadowing primaire.

8.3.3. Shadowing Secondaire - Secondary Shadowing.

La politique de shadowing secondaire permet à un DSA consommateur d'information répliquée de jouer le rôle d'un DSA fournisseur de cette même information avec d'autres DSAs consommateurs n'ayant pas d'accord de shadowing (shadowing agreement) avec le DSA maître de cette unité de réplication. L'accord de shadowing sera expliqué plus longuement dans la suite de ce chapitre (cfr 8.4.1.).

Signalons donc que dans ce cas, certains DSAs consommateurs d'information répliquée ont un accord de shadowing avec le DSA maître et d'autres pas. Cependant ces autres DSAs, contenant la même unité de réplication ont un accord de shadowing avec des DSAs consommateurs, en accord avec le DSA maître pour cette unité d'information répliquée, qui jouent donc ici le rôle de fournisseur de cette information.

Cependant, aucun des DSAs consommateurs de cette unité de réplication n'a le droit de modifier celle-ci. De la même façon que lors de la politique primaire de shadowing, les opérations de modification sont redirigées vers le DSA maître.

En ce qui concerne l'envoi des mises-à-jour, ce sont les DSAs fournisseurs de l'information répliquée qui s'en chargent et cela qu'ils soient maîtres ou également consommateurs de cette information répliquée.

La figure 8.2. represente un exemple de cette politique de shadowing secondaire.

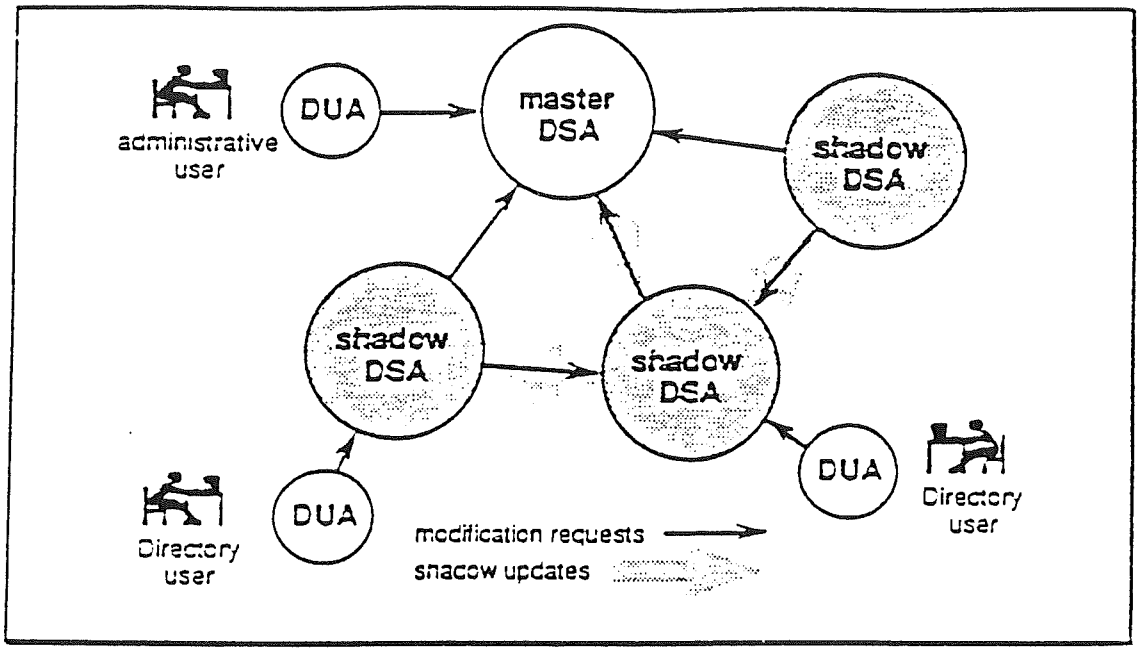


Fig 8.2. : Politique de shadowing secondaire.

8.4. Aperçu Général du Service de Réplication du Directory.

Le service de réplication (shadow service) défini ici fournit au Directory un mécanisme standard qui permet et supporte l'emploi d'information répliquée.

Afin de pouvoir employer un tel service, les administéchoueurs de deux DSAs doivent d'abord atteindre un accord de shadowing, accord selon lequel le service de réplication prendra place. Cet accord et ses spécifications techniques sont discutés au paragraphe 8.4.1..

Nous détaillerons ensuite la manière dont l'information répliquée est représentée au paragraphe 8.4.2.. Nous introduirons ensuite, au paragraphe 8.4.3. la façon selon laquelle le transfert de l'information répliquée du DSA fournisseur aux DSAs consommateurs prend place.

8.4.1. L'Accord de Shadowing - Shadowing Agreement.

L'accord de shadowing qui est atteint par les adminitéchoueurs de deux DSAs forme les bases techniques et politiques pour l'emploi du service de réplication du Directory. Cet accord peut contenir n'importe quel ensemble de termes acceptables pour les deux parties tels que des conditions sur l'effacement de l'information répliquée, telle qu'une politique de sécurité...

Certains aspects de l'accord doivent absolument être spécifiés. Par exemple un identifiant par lequel l'accord sera référencié doit être défini, une spécification de l'unité de réplication doit être donnée, une information relative au systême de mise-à-jour et optionnellement le point d'accès du DSA maître de l'information répliquée doivent être définis.

8.4.1.1. Aspects Techniques de l'Accord de Shadowing.

Les autorités administratives des DSAs peuvent exiger qu'un accord bilatéral soit établi avant que le shadowing puisse prendre place. Cet accord peut être exigé dans le cas où les deux DSAs se trouvent dans deux domaines différents.

Un accord de shadowing est exigé avant que l'information répliquée puisse être partagée entre deux DSAs. Ceci établi, les paramètres techniques de l'accord spécifient la fréquence des mises-à-jour, l'étendue à répliquer ainsi que l'information à répliquer.

8.4.1.1.1. Spécification de l'Accord de Shadowing.

L'accord de shadowing est spécifié de la manière suivante :

- par un identifiant unique indiquant de quel accord il s'agit.
- par un ensemble d'informations sur l'accord de shadowing composé de :

  - d'un sous arbre comportant les entrées et les attributs à répliquer (unit of replication).
  - du mode de mise-à-jour.
  - du point d'accès au DSA maître de l'unité de réplication.
  - d'un booléen indiquant la stratégie de mise-à-jour (totale ou partielle).

Les deux sous-points suivants détaillent l'unité de réplication et le mode de mise-à-jour.

8.4.1.1.2. L'Unité de Réplication.

Cette section décrit comment les portions du DIT peuvent être répliquées en définissant la granularité de l'information du DIT qui peut être répliquée. Plus simplement, les mécanismes de réplication au sein du Directory sont basés sur la définition d'un sous-ensemble du DIT qui sera répliqué. Ce sous-ensemble porte le nom d'unité de réplication.

Etant donné le fait que le mécanisme de réplication est seulement défini entre deux DSAs, l'information à répliquer doit être contenue complétement au sein d'un même DSA.

L'unité de réplication est définie en trois parties. Une partie définissant l'étendue à répliquer, une autre définissant les attributs à répliquer au sein de cette étendue et une troisième spécifiant les connaissances subordonnées optionnelles. Ces trois partie seront détaillées dans la suite de cette section.

8.4.1.1.2.1. Spécification de l'Etendue - Area Specification.

L'étendue définit la zone à répliquer. Ceci inclut le nom distingué du point administratif pour cette zone ainsi que la spécification d'un sous-arbre relatif à ce point administratif. De manière générale, un sous-arbre définit l'étendue à répliquer mais il est permit de raffiner ce sous-arbre afin d'éliminer des parties de celui-ci. Ce raffinement s'éffectue en deux phases qui sont détaillées ci-dessous (filtrage).

8.4.1.1.2.1.1. Spécification du sous-arbre.

La première phase sert essentiellement à spécifier la forme du sous-arbre qui servira de base à la réplication au sein d'un DSA. Ceci est réalisé en définissant le corps de l'arbre en se basant sur deux éléments :

- la base qui est utilisée pour fournir le nom de l'objet de base (base object) de l'unité de réplication relative à un point administratif approprié. Plus simplement, le nom de l'objet de base est le nom de la racine du sous-arbre.
- l'ossature (chop) qui est utilisé pour définir le sous-arbre en lui-même en employant une technique de profondeur maximum et d'exclusion. Si ce paramètre est absent, le sous-arbre est donc l'arbre "entier" ayant pour racine la base.

8.4.1.1.2.1.2. Raffinement du sous-arbre.

Lors de la deuxième phase de raffinement, on applique un filtre sur le sous-arbre sélectionné. Le filtrage ne peut avoir lieu que sur les classes d'objets ou sur les sous-entrées.

Il est important de noter que le filtrage peut conduire à un arbre qui n'en est plus un. Pour de tels arbres, des DSEs de type glue doivent être rajoutés pour chacune des entrées manquantes utiles à la reconstruction d'un "vrai" arbre.

8.4.1.1.2.2. Sélection des Attributs - Attributes Selection.

Ce paramètre de sélection des attributs permet de définir l'ensemble des attributs utilisateurs ou opérationnels qui doivent être répliqués au sein de l'unité de réplication.

Nous n'allons par rentrer dans les détails ici, mais signalons simplement le principe général de cette sélection. Il s'agit en fait d'énumérer pour chaque classe d'objets ou de sous-entrées, l'ensemble des classes d'attributs devant être répliquées. Ensuite, pour chaque classe d'attributs, il faut spécifier quels sont ceux qui doivent faire partie de l'unité de réplication.

8.4.1.1.2.3 Connaissances Subordonnées - Subordinate Knowledge.

Le dernier élément requis pour la spécification de l'unité de réplication contient les connaissances subordonnées. Ces connaissances peuvent être composées de contexte de dénomination, de références spécifiques ou non. Nous ne nous attarderons pas sur ce sujet. Notons simplement qu'elles sont facultatives.

8.4.1.1.3. Le Mode de Mise-à-jour.

L'argument de mode de mise-à-jour inclus dans l'accord de shadowing spécifie quand et comment auront lieu ces dites mises-à-jour. Cet argument est composé de deux éléments :

- le mode de mise-à-jour du fournisseur.
- le mode de mise-à-jour du consommateur.

Malgré le fait que ces deux composants soient présents dans ce paramètre, un choix doit être fait en ce qui concerne le mode de mise-à-jour (par le fournisseur ou par le consommateur).

Dans le cas de la mise-à-jour initiée par le fournisseur, on peut choisir si ces mises-à-jour auront lieu lors de chaque modification d'un élément de l'unité de réplication ou si elles se feront périodiquement.

Dans le cas de la mise-à-jour demandée par le consommateur, il faut simplement spécifier la période de mise-à-jour.

8.4.2. L'Information Répliquée - Shadowed Information.

L'information répliquée est un ensemble logique d'information qui est recopié chez le DSA consommateur de cette information. L'information répliquée est composée de trois éléments :

- étendue (area information) : information au sujet des DSEs dont les noms appartiennent à cette étendue répliquée.
- préfixe (prefix information) : information permettant de savoir où se trouve l'étendue répliquée dans le DIT. En fait cette information de préfixe contient le chemin menant du DSE racine à la racine de l'étendue répliquée.
- information subordonnée (subordinate information) : il s'agit de références subordonnées à l'étendue répliquée.

La figure 8.3. illustre les différents composants de l'information répliquée. Ces composants seront discutés dans la suite de cette section.

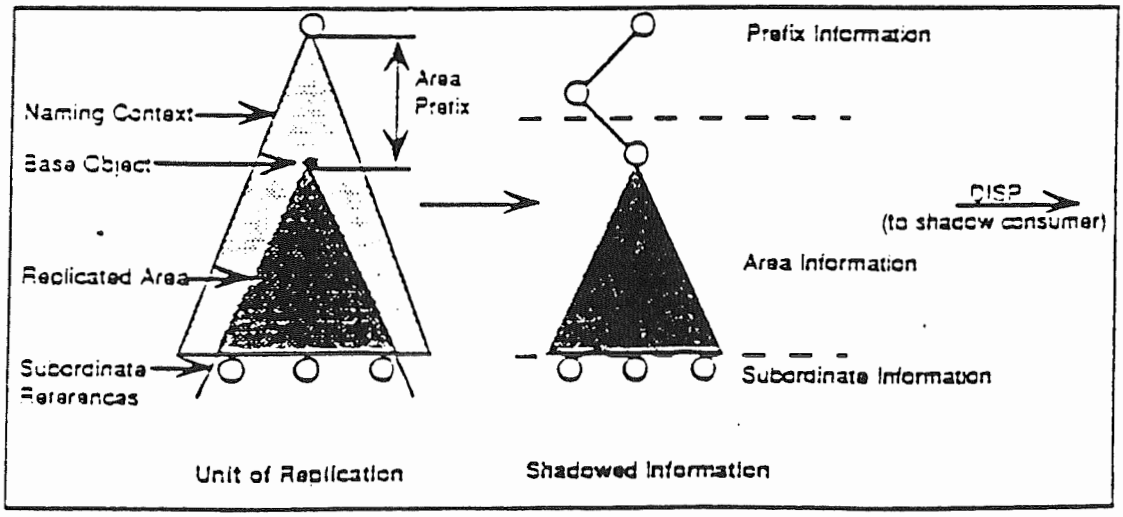


Fig 8.3. : Composants de l'information répliquée.

On notera, comme l'illustre la figure ci-dessus, que l'étendue répliquée doit être entièrement contenue dans un contexte de dénomination unique.

L'information répliquée est conceptuellement fabriquée à partir de DSEs répliqués (SDSEs) qui sont détaillés au point 8.4.2.1.. Les composants de l'information répliquée sont détaillés au point 8.4.2.2..

8.4.2.1. Shadowed DSA Specific Entry - SDSE.

Un shadowed DSE (SDSE) est une information associée à un nom spécifique au sein d'une information répliquée. Il représente l'information d'un DSE à répliquer. Les SDSEs sont des entités conceptuelles qui facilitent la spécification et la modélisation de l'information qui doit être répliquée.

Un SDSE est analogue à un DSE et est constitué de :

- un type de SDSE obligatoire.
- des attributs utilisateurs (dérivés de l'information contenue dans une entrée pour les DSEs correspondants aux entrées devant être répliquées.
- attributs opérationnels.
- subordinate-completeness flag.
- attribute-completeness flag.

En ce qui concerne le type d'un SDSE, ce type est le même que pour un DSE, mais il a plusieurs options : glue, context, rank-and-file, suborbordinate reference, non-specific subordinate reference, administrative point et subentry.

Le subordinate-completeness flag est un booléen qui est uniquement présent pour les SDSEs compris dans une étendue répliquée. Quand il est présent, il a la signification suivante :

Le drapeau (flag) a la valeur vraie seulement si une des conditions suivantes est respectée pour un SDSE particulier :

- le SDSE représente une entrée feuille (leaf entry).
- l'information répliquée contient un SDSE pour chaque entrée subordonnée ou pour chaque référence subordonnée (spécifique ou non) connue par le DSA maître.

Le attribute-completeness flag est un booléen dont la valeur est vraie si tous les attributs utilisateurs de l'entrée sont présents dans le SDSE. Ce drapeau n'est présent que pour les SDSEs contenant l'information d'une entrée.

8.4.2.2. Composants de l'Information Répliquée.

L'information répliquée contient trois types d'information de base, comme nous l'avons d'ailleurs dit précédemment.

8.4.2.2.1. L'Information de Préfixe.

Si l'étendue répliquée ne débute pas immédiatement en dessous de la racine du DIT, l'information répliquée contiendra des SDSEs pour chaque entrée faisant partie du préfixe de l'étendue répliquée (le chemin menant de la racine du DIT jusqu'à la racine de l'étendue répliquée). Chacun de ces SDSEs sera du type glue et représentera uniquement le RDN de l'entrée, sauf si il représente un point administratif contenant de l'information sur la politique administrative de ce point. Le DSE racine est représenté par un SDSE vide.

Dans le cas des points administratifs, le type du SDSE n'est plus glue mais admPoint et contient tous les attributs utiles pour l'étendue répliquée. Dans le cas des sous-entrées (subentries), le type de SDSE choisi est le type subEntry.

8.4.2.2.2. L'Information sur l'Etendue.

Toutes les entrées dans l'arbre du DSA consommateur qui sont inclues dans l'étendue répliquée sont représentée sous la forme de SDSEs de type rf. Ces SDSEs contiennent les attributs de l'entrée comme sélectionné par le paramètre de sélection des attributs lors de l'accord de shadowing. Les attributs contenus dans les sous-entrées sont sélectionnés de la même manière et ces sous-entrées sont représentées par des SDSEs de type subEntry. Si un attribut au moins a été sélectionné, l'attribut de classe d'objet (ObjectClass attribute) et l'information de contrôle d'accès seront inclus dans le SDSE de cette entrée. Le drapeau de complétude des attributs (attribute completeness flag) sert à indiquer si tous les attributs utilisateurs, présents dans le DSE sont présents dans le SDSE. Les attributs opérationnels ne sont jamais inclus au sein d'un SDSE.

Si le DSE est de type admPoint, le SDSE sera du type additionnel admPoint. Si le DSE est de type cp, le SDSE correspondnat le sera aussi.

Si un filtre a été appliqué lors de la création de l'unité de réplication, il se peut que des "trous" apparaissent dans la structure d'arbre de cette unité. Pour chacune des entrées qui a été retirée par filtrage, les règles suivantes sont applicables :

- s'il existe des SDSEs subordonnés à cette entrée qui n'ont pas été retirés par filtrage, un SDSE de type glue est rajouté pour cette entrée.
- dans le cas contraire, càd pas de SDSE subordonnés, le drapeau de complétude des subordonnés (subordinate-completeness flag) du SDSE correspondant à l'entrée immédiatement supérieure à l'entrée supprimée est mis à faux et le SDSE de l'entrée supprimée ne fait pas partie de l'information répliquée.

8.4.2.2.3. L'Information Subordonnée.

Le type d'information subordonnée requis est spécifié lors de l'accord de shadowing. Si une connaissance subordonnée (subordinate knowledge) ou une connaissance étendue (extended knowledge) est requise, des références subordonnées sont inclues au titre de SDSEs de type subr ou nssr, selon le cas, et complétées de l'information adéquate.

Des SDSEs de type glue doivent être inclus afin de maintenir la connection avec les SDSEs de l'étendue répliquée.

8.4.3. Les Opérations de Shadowing.

L'information répliquée est transmise du DSA fournisseur au DSA consommateur grâce aux opérations de shadowing. Ces opérations fournissent deux modèles fondamentalement différents pour la mise-à-jour de l'information répliquée :

- le "push" modèle où la mise-à-jour est initiée par le fournisseur.
- le "pull" modèle où la mise-à-jour est demandée par le consommateur.

Nous ne détaillerons pas ces deux modèles. Cependant, dans chacun de ceux-ci, le transfert de l'information peut prendre une de ces deux formes :

- transfert total : l'ensemble de l'information répliquée est transmise.
- transfert partiel : seulement la partie modifiée de l'information répliquée est transmise.

Trois opérations de shadowing ont été définies à ce stade :

- l'opération de mise-à-jour coordonnée (CoordinateShadowUpdate) qui est utilisée dans le "push" modèle afin de permettre au fournisseur de l'information répliquée d'indiquer l'accord de shadowing pour lequel il a l'intention d'envoyer une mise-à-jour, ou pour indiquer la date de la dernière mise-à-jour pour cet accord ou pour indiquer la stratégie de mise-à-jour (totale ou partielle).
- l'opération de mise-à-jour (update shadow)qui est utilisée par le fournisseur d'information répliquée dans le cas d'une réponse positive à la précédente opération. Cette opération permet d'envoyer l'information répliquée ou les modifications de celle-ci selon la stratégie choisie.
- l'opération de demande de mise-à-jour (request shadow update) qui est utilisée dans le "pull" modèle par le consommateur d'information répliquée. Cette opération permet d'indiquer d'une part l'accord de shadowing pour lequel il désire recevoir une mise-à-jour, d'autre part pour indiquer la stratégie de mise-à-jour et la date de dernière mise-à-jour. Dans le cas d'une réponse positive, le fournisseur de l'information répliquée envoie ces dites mises-à-jour.

8.4.3.1. Les Ports du Service de Réplication.

Les DSAs fournissent les services de réplication aux autres DSAs via les ports de service de mise-à-jour de l'information répliquée (Shadow Update Service Port). Les DSAs peuvent aussi utiliser les ports de service d'un autre type : Operational Binding Management Service Ports pour établir et terminer un accord de shadowing.

Ces ports sont illustrés par la figure suivante.

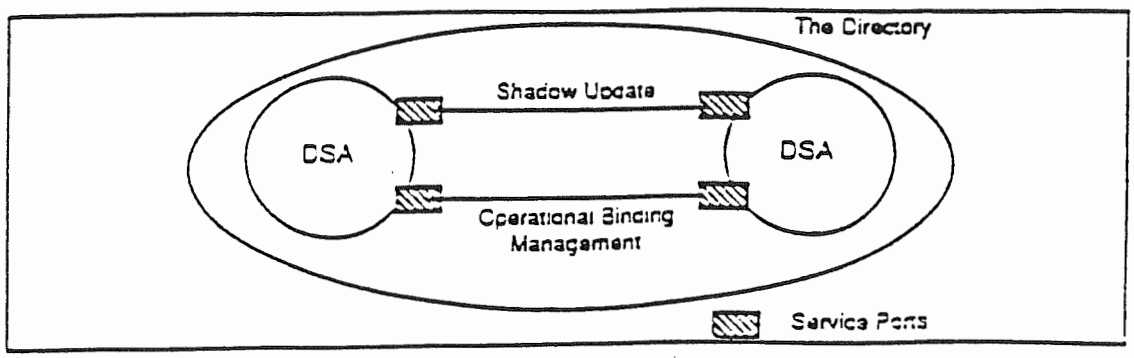


Fig 8.4. : Les ports du service de réplication.

Le port de service de mise-à-jour sont utilisés pour toutes les opérations associées au DISP (Directory Information Shadow Protocol).

L'opération DSAShadowBind permet au DSA de connecter son port de mise-à-jour à celui de n'importe quel autre DSA dans le but de shadowing. L'opération DSAShadowUnbind permet de déconnecter une paire de DSAs.

8.4.3.2. Directory Information Shadow Protocol - DISP.

Lors de cette section, nous allons définir quelles sont les opérations du Directory Information Shadow Protocol (DISP) qui sont utilisées par les fournisseurs et les consommateurs d'information répliquée.

8.4.3.2.1. L'Opération Coordinate Shadow Update.

L'opération de mise-à-jour coordonnée est utilisée par le fournisseur d'information répliquée pour indiquer l'accord de shadowing pour lequel il a l'intention d'envoyer des mises-à-jour.

Cette opération a pour paramètres arguments les éléments suivants pour chaque accord de shadowing :

- agreementID qui identifie un accord de shadowing.
- lastUpdateSent qui indique la date de la dernière mise-à-jour pour cet accord.
- updateStéchouegy qui identifie la stratégie de mise-à-jour que le fournisseur à l'intention d'utiliser. Il existe deux types de stratégies possibles :

- standard : le fournisseur peut choisir une des trois options suivantes :

  - none : pas de modification.
  - partielle : remplacement de l'information modifiée.
  - totale : remplacement complet de l'unité de réplication.

- none : cette option est utilisée quand le fournisseur désire informer le consommateur qu'aucune modification n'est survenue depuis la dernière mise-à-jour.

Si l'opération réussit, un résultat vide est renvoyé par le consommateur. Dans le cas d'un échec, une erreur est retournée au fournisseur. Nous ne nous attarderons pas sur les différents types d'erreur.

8.4.3.2.2. L'Opération Update Shadow.

Une opération de mise-à-jour des informations répliquées est invoquée par un DSA fournisseur afin d'envoyer des mises-à-jour à un DSA consommateur d'une unité de réplication. Avant d'utiliser cette opération, une opération CoordinateShadowUpdate a dû être envoyée et réussie.

Cette opération Update Shadow a les paramètres arguments suivants :

- agreementID qui est un entier identifiant l'accord de shadowing.
- updateTime qui est une date fournie par le DSA fournisseur et qui sera utilisée lors de la prochaine opération de mise-à-jour (CoordinateShadowUpdate ou RequestShadowUpdate) afin d'assurer que les deux DSAs, fournisseur et consommateur ont une vue commune sur l'information répliquée.
- updateWindow qui est un paramètre optionnel qui indique au DSA consommateur le laps de temps maximum avant l'envoi de la prochaine mise-à-jour.
- updateInfo qui représente l'information nécessaire au DSA consommateur pour remettre à jour son information répliquée. Cette information peut être une copie totale de l'unité de réplication ou seulement une partie de celle-ci.

Voyons maintenant plus en détail de quoi est constitué le paramètre updateInfo. En ce qui concerne cette information, trois cas sont possibles (TotalRefresh, IncrementalRefresh, NoRefresh).

8.4.3.2.2.1. NoRefresh.

Dans ce cas-ci, l'updateInfo a la valeur NULL ce qui signifie qu'il ny a eu aucun changement à l'information répliquée depuis la dernière mise-à-jour. Cette option est utilisée dans le cas où le DSA fournisseur se doit d'envoyer les mises-à-jour à intervalles réguliers.

8.4.3.2.2.2. TotalRefresh - Total.

L'option "Total" fournit une nouvelle instance de l'information répliquée. Celle-ci est représentée sous la forme de sous-arbres en utilisant un algorithme de recherche en profondeur d'abord.

L'information répliquée complète est donnée en commençant à partir de la racine et en comprenant tous les SDSEs de l'arbre. L'argument TotalRefresh est composé d'une séquence de deux paramètres :

- sDSE qui est un ensemble composé de :

  - sDSEType qui représente le type de SDSE en question.
  - subComplete qui est un booléen indiquant si oui ou non les connaissances subordonnées sont complètes.
  - attComplete qui est un autre booléen indiquant si tous les attributs utilisateurs sont inclus ou non.

- subTree qui est un ensemble de sous-arbres composés d'une séquence de noms relatifs distingués et de l'information proprement dite.

On notera que l'absence de certains SDSEs anciennement contenus dans l'information répliquée signifie qu'ils ont été effacés.

8.4.3.2.2.3. IncrementalRefresh - Partiel.

L'option "partiel" fournit, au lieu d'un remplacement complet de l'information répliquée, seulement les changements qui sont apparus au sein de cette information depuis la dernière mise-à-jour.

L'argument IncrementalRefresh est composé d'une séquence des deux paramètres suivants :

- sDSEChanges qui vaut soit add, soit remove, soit modify dont la signification est la suivante :

  - add qui fournit une copie complète d'un SDSE.
  - remove qui indique que le SDSE ne sera plus présent dans l'information répliquée.
  - modify qui comprend les changements à effectuer à un SDSE.

Ces changements sont représentés par la séquence suivante :

- rename qui est utilisé pour modifier le RDN d'un SDSE.
- un des deux paramètres suivants :

  - replace qui contient l'ensemble des attributs du SDSE.
  - changes qui contient seulement les attributs modifiés du SDSE.

- SDSEType qui indique le type du SDSE.
- Subcomplete et attComplete qui ont la même signification que précédemment.

- subordinateUpdates, utilisé pour modifier ou créer des subordonnés, qui est ensemble de SubordinateChanges qui est composé de la séquence suivante :

  - RDN.
  - l'information proprement dite.

L'opération Update Shadow fournit un résultat qui est soit une erreur si l'opération échoue, soit un résultat "vide" si elle réussit.

8.4.3.2.3. L'Opération Request Shadow Update.

L'opération RequestShadowUpdate est utilisée par le DSA consommateur de l'information répliquée afin de demander l'envoi des mises-à-jour au DSA fournisseur.

Cette opération a pour argument un séquence de paramètres dont voici la signification :

- agreementID qui est un entier identifiant l'accord de shadowing.
- lastUpdateReceived qui indique la date de la dernière mise-à-jour.
- requestStéchouegy qui identifie le type de stratégie qui sera employé (partiel, total).

Si cette opération réussit, le DSA fournisseur renvoie un résultat qui est une séquence composée de :

- updateTime.
- updateWindow.
- updateInfo.

Ces paramètres ont la même signification que ceux utilisés pour l'opération UpdateShadow (cfr 8.4.3.2.2.).

Si l'opération échoue, une erreur est renvoyée.

9. Exemple de Modélisation des Connaissances. [X.500]

L'exemple suivant illustre un hypothétique DIT distribué en trois DSAs, ainsi que l'information que les DSAs doivent maintenir pour supporter cette distribution.

Les figures suivantes utilisent les symboles définis ci-dessous.

- ● Entrée objet
- ▲ Entrée alias
- ■ Sous-entrée
- ○ Etendue administrative autonome
- Contexte de dénomination

La première de ces figures illustre le DIT hypothétique. Il est partitionné en quatre étendues administratives autonomes : les cas dégénérés des entrées {C=VV} et {C=WW} et les deux sous-arbres de racine {C=WW, O=ABC} et {C=VV, O=DEF}. Une entrée de nom distingué {C=VV, O=DEF, OU=K} est un alias dont l'entrée réelle est {C=WW, O=ABC, OU=I}.

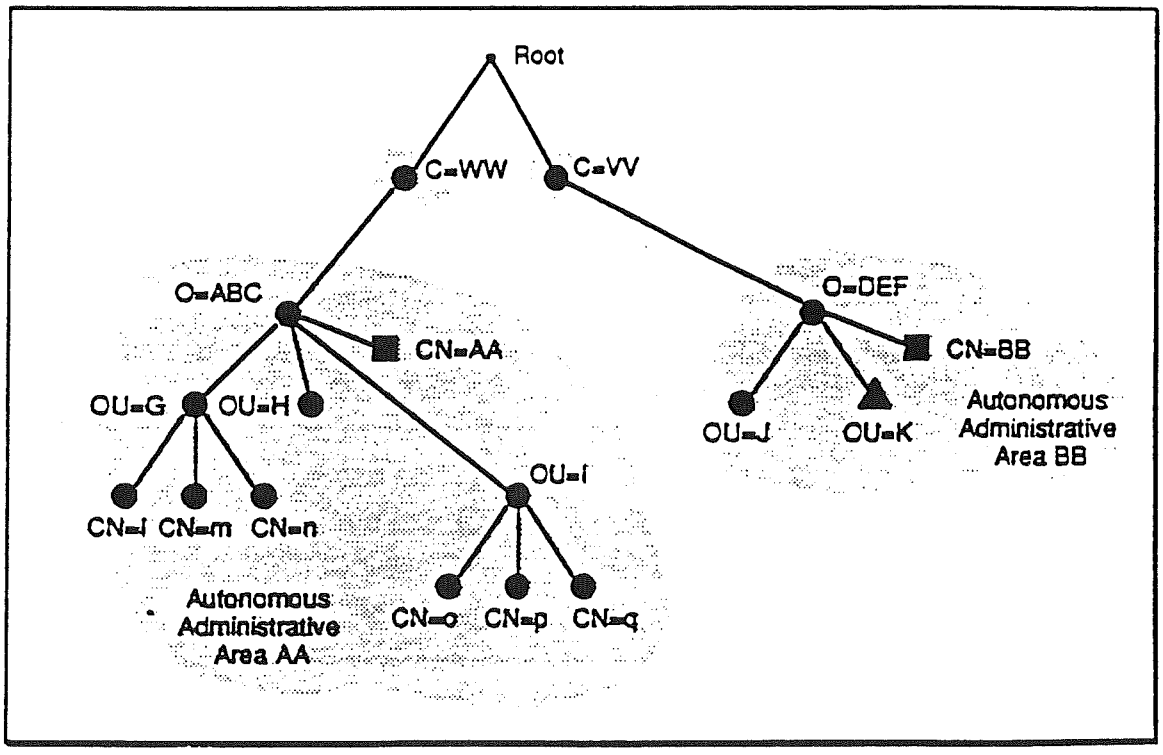


Fig 9.1. : Un hypothétique DIT.

La deuxième figure illustre la distibution du DIT en cinq contextes de dénomination (A, B, C, D et E) et leur stockage en trois DSAs (DSA1, DSA2 et DSA3). Dans cet exemple, le DSA1 maintient le contexte de dénomination C; le DSA2, les contextes A, B et E; le DSA3, le contexte D.

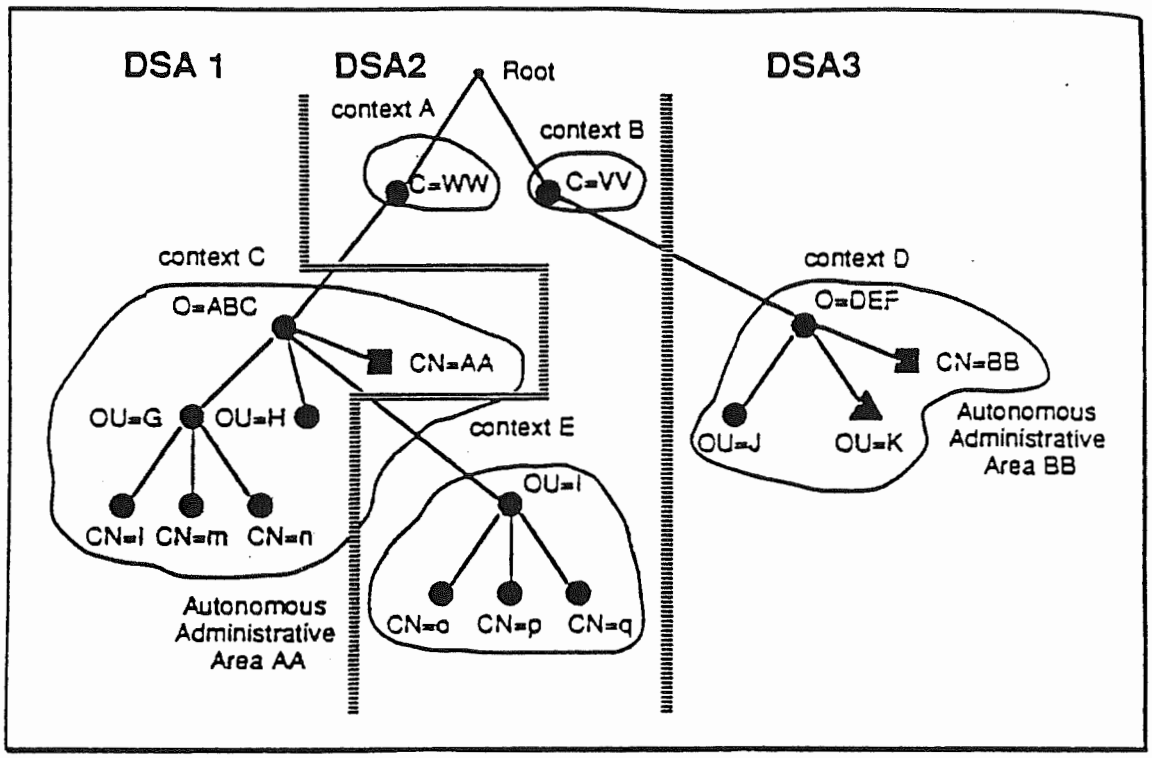


FIG 9.2. : Un hypothétique DIT distibué en trois DSAs.

Les connaissances maintenues par les trois DSAs sont les suivantes :

- le DSA1 emploie le DSA2 comme référence supérieure et a une référence subordonnée non-spécifique vers le DSA2.
- le DSA2 est un DSA de premier niveau et maintient une référence subordonnée vers le DSA1 pour le contexte C et une référence supérieure immédiate vers celui-ci pour le contexte immédiatement supérieur au contexte E. Le DSA2 maintient aussi une référence subordonnée vers le DSA3 pour le contexte de dénomination D.
- le DSA3 emploie aussi le DSA2 comme référence supérieure et il maintient une référence croisée vers le DSA1 pour le contexte de dénomination C.

Les figures suivantes illustrent l'information contenue dans chaque DSA afin de supporter cette configuration. On y utilise les symboles suivants.

● DSE entrée	⊜ DSE racine
▲ DSE alias	○ DSE glue
■ DSE sous-entrée	▽ DSE subr
(x) DSE de type x	⊠ DSE xr

Comme le DSA1 n'est pas un DSA de premier niveau, son DSE racine contient une référence supérieure, qui dans cet exemple, est le point d'accès du DSA2. Le type de ce DSE est racine + supr.

Le DSA1 maintient un DSE glue afin de représenter la connaissance du nom {C=WW}.

L'étendue administrative autonome AA est subdivisée en deux contextes de dénomination C et E, avec le contexte C tenu par le DSA1. Dans un but de simplification de cet exemple, on supposera que les étendues administratives relatives au contrôle d'accès et à l'information de sous-schéma coïncident et qu'il n'y a qu'un seul domaine de contrôle d'accès et un seul sous-schéma pour l'étendue administrative autonome toute entière.

Pour le DSA1, le DSE se trouvant à {C=WW, O=ABC} représente un point administratif pour AA, le préfixe du contexte de dénomination C et une référence subordonnée non-spécifique au DSA2. Ce DSE est donc du type entrée + cp + admPoint + nssr. L'information opérationnelle concernant cette étendue est maintenue dans la sous-entrée de nom {C=WW, O=ABC, CN=AA}.

Le DSA1 maintient aussi les entrées contenues dans le contexte de dénomination C : {C=WW, O=ABC, OU=G}, {C=WW, O=ABC, OU=H}, {C=WW, O=ABC,OU=G, CN=i}, {C=WW, O=ABC, OU=G, CN=m}, {C=WW, O=ABC, OU=G, CN=n}.

Finalement, le DSA1 maintient en {C=WW, O=ABC} un DSE pour une référence subordonnée non-spécifique contenant l'information de point d'accès pour le DSA2.

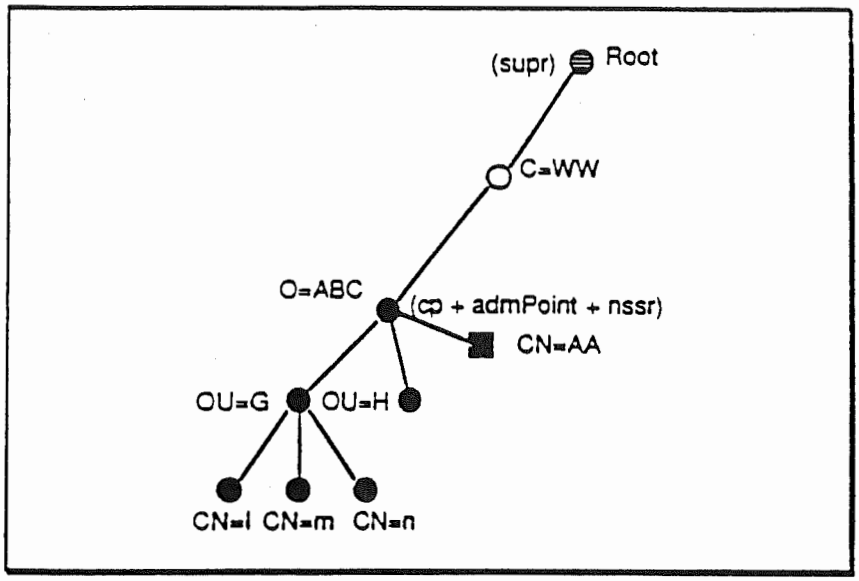


Fig 9.3. : L'arbre d'information des DSAs pour le DSA1.

Dans cette situation hypothétique, le DSA2 est un DSA de premier niveau. Son DSE racine ne contient donc pas de référence supérieure.

Les deux étendues administratives dégénérées, {C=WW} et {C=VV} sont représentées par des DSES de type cp + entrée + admPoint.

Les connaissances subordonnées du DIT sont représentées par deux DSEs de référence subordonnée, {C=WW, O=ABC} et {C=VV, O=DEF}. Dans le premier cas, ce DSE est du type subr + admPoint + immSupr pour des raisons qui seront décrites plus tard.

Sur la figure suivante, le DSA2 est configuré de telle manière qu'une seule sous-entrée maintient l'information opérationnelle de l'étendue AA. Ceci requiert qu'une copie de la sous-entrée soit présente dans le DSA2 (pour des raisons de performance aussi). Une façon de réaliser ceci est d'établir un lien entre le DSA1 et le DSA2 afin de maintenir une copie de la sous-entrée. Dans ce cas, l'information opérationnelle concernant cette étendue se trouve dans un DSE, dont le nom est {C=WW, O=ABC, CN=AA}, qui est du type sous-entrée + shadow.

Finalement, le contexte de dénomination E est maintenu par le DSE {C=WW, O=ABC, OU=I} qui est du type cp + entrée et les trois DSEs de type entrée {C=WW, O=ABC, OU=I, CN=o}, {C=WW, O=ABC, OU=I, CN=p} et {C=WW, O=ABC, OU=I, CN=q}.

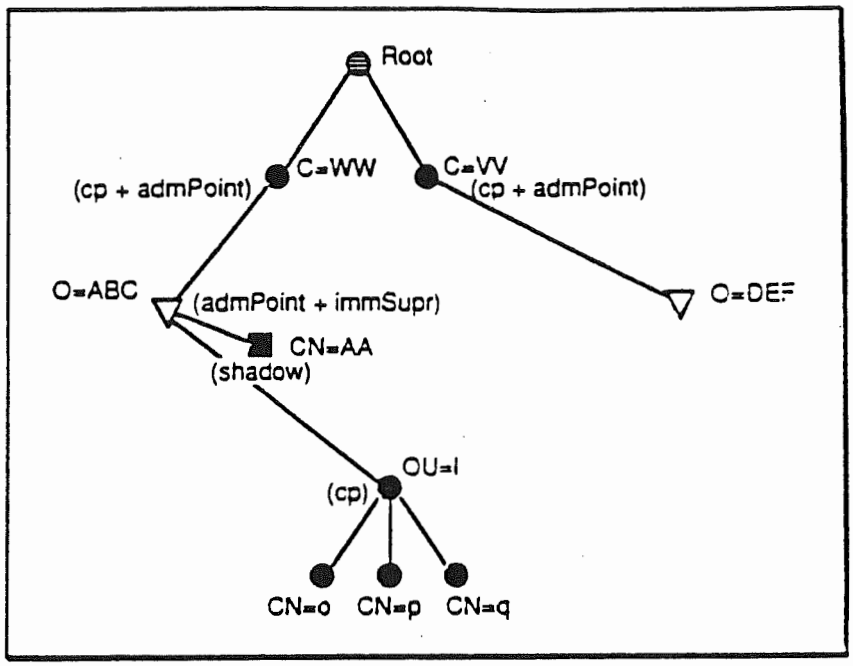


Fig 9.4. : L'arbre d'information des DSAs pour le DSA2.

Une autre façon de configurer le DSA2 est illustré à la figure suivante.

Celle-ci diffère de la configuration illustrée à la figure précédente seulement dans la façon de maintenir l'information opérationnelle de l'étendue.

La stratégie employée ici est de partionner AA (i.e. partition du domaine de l'information de contrôle d'accès et de façon similaire, de l'information du sous-schéma) en deux étendues administratives autonomes, une coincidant avec le contexte de dénomination C et l'autre avec le contexte de dénomination E.

Dans ce cas, le DSE de préfixe de contexte {C=WW, O=ABC, OU=I} devient aussi un point administratif ; le type de ce DSE est donc cp + admPoint + entry. L'information opérationnelle de l'étendue "réduite" est tenue par la sous-entrée {C=WW, O=ABC, OU=I, CN=AA}.

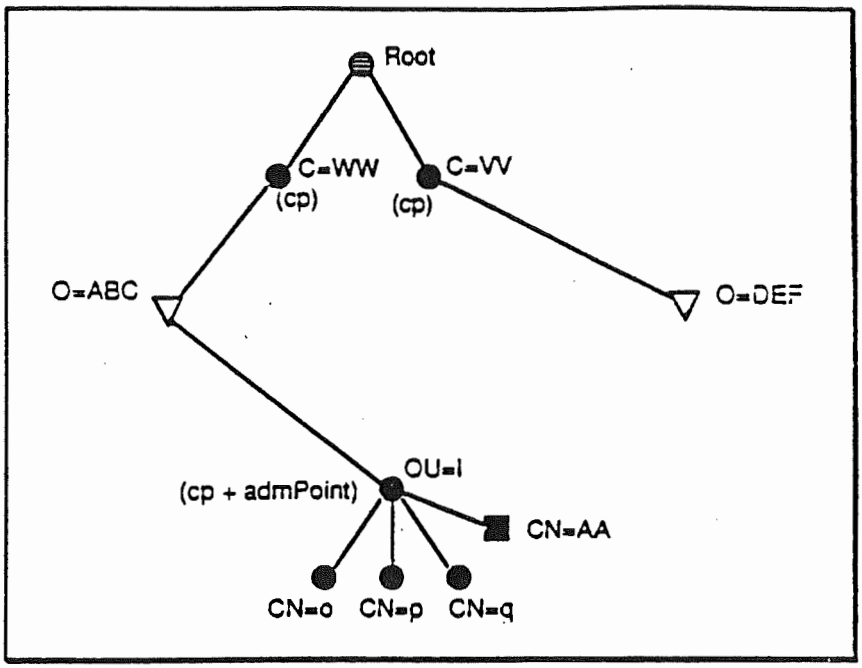


Fig 9.5. : Un arbre alternatif de l'information du DSA2.

La figure suivante illustre l'arbre d'information du DSA3.

Comme pour le DSA1, le DSA3 n'est pas un DSA de premier niveau. Son DSE racine contient une référence supérieure, qui dans cet exemple est le point d'accès au DSA2. Le type de ce DSE est racine + supr.

Le DSA2 maintient aussi un DSE de type glue pour représenter la connaissance du nom {C=VV}.

L'étendue administrative autonome BB coincide avec le contexte de dénomination D. Dans un but de simplification, comme dans le cas de l'étendue administrative AA, l'étendue administrative relative au contrôle d'accès et au sous-schéma coincident et qu'il n'y a qu'un seul domaine de contrôle d'accès et qu'un seul sous-schéma pour l'étendue administrative tout entière.

Pour le DSA3, le DSE situé à {C=VV, O=DEF}, représentant le point administratif de BB et le préfixe du contexte de dénomination D, est du type entrée + cp + admPoint. L'information opérationnelle de cette étendue est contenue dans la sous-entrée {C=VV, O=DEF, CN=BB}.

Le DSA3 maintient aussi une entrée objet et une entrée alias. En ce qui concerne l'alias, dans cette entrée on retrouve le nom de l'entrée réelle {C=WW, O=ABC, OU=I}.

Finallement, le DSA3 maintient une référence croisée vers le contexte de dénomination C. Cette information se trouve dans un DSE de type xr et de nom {C=WW, O=ABC}.

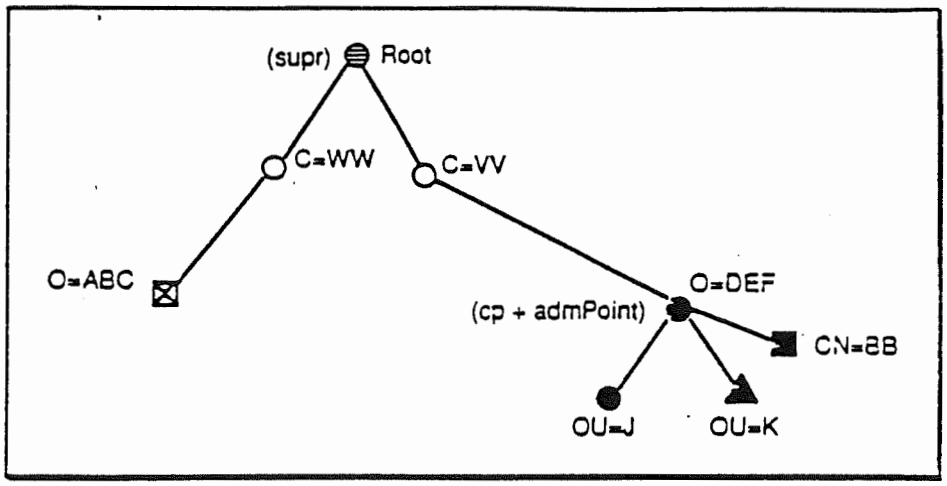


Fig 9.6. : L'arbre d'information du DSA3.

10. Bibliographie.

Recommandation X.500 et ISO 9594-1, "The Directory - Overview of concepts, Models and services", Décembre 1988.

Recommandation X.501 et ISO 9594-2, "The Directory - Models", Décembre 1988.

Recommandation X.509 et ISO 9594-8, "Information Processing Systems - Open Systems Interconnection - The Directory - Authentication Framework", Décembre 1988.

Recommandation X.511 et ISO 9594-3, "Information Processing Systems - Open Systems Interconnection - The Directory - Abstract Service Definition", Décembre 1988.

Recommandation X.518 et ISO 9594-4, "Information Processing Systems - Opens Systems Interconnection - The Directory - Procedures for Distributed operation", Décembre 1988.

Recommandation X.519 et ISO 9594-5, "The Directory - Protocol Specifications", Décembre 1988.

Recommandation X.520 et ISO 9594-6, "Information Processing systems - Open Systems Interconnection - The Directory - Selected Attribute Types", Décembre 1988.

Recommandation X.521 et ISO 9594-7, "Information Processing systems - Open Systems Interconnection - The Directory - Selected Objects Classes", Décembre 1988.

ISO/IEC 9594-2 / 3rd PDAM-1 for Access Control, "Proposed Revisions to Recommendation X.501 for Access Control", Juin 1991.

ISO/IEC 9594-2 / 2nd PDAM-2 for Schema, "Proposed Revisions to recommendation X.501 for Schema", Juin 1991.

ISO/IEC 9594-3 / 3rd PDAM-1 for Access Control, "Proposed Revisions to Recommendation X.511 for Access Control", Juin 1991.

ISO/IEC 9594-4 / 3rd PDAM-1 for Access Control, "Proposed Revisions to Recommendation X.518 for Access Control", Juin 1991.

ISO/IEC 9594-1 / 2nd PDAM-1 for Replication Schema and Access Control, "Proposed Revisions to Recommendation X.500", Juin 1991.

"Meeting Report from Phoenix Meeting", Avril et Mai 1991.

ISO/IEC 9594-7 / 2nd PDAM-1 for Schema Extensions, " Proposed Revisions to Recommendation X.521", Juin 1991.

ISO/IEC 2nd CD 9594-9 for Replication, " Draft Recommendation X.5rp", Juin 1991.

ISO/IEC 9594-2 / 2nd PDAM-3 for Replication, " Proposed Revisions to Recommendation X.501 for Replication", Juin 1991.

ISO/IEC 9594-4 / 2nd PDAM-2 for Replication, Schema and Enhanced Search, "Proposed Revisions to Recommendation X.518 for Replication, Schema and Enhanced Search", Juin 1991.

"The Little Black Book", Marshall T. Rose, Prentice Hall Series in Innovative Technology, ???.

"EDI Use of X.500 Directory", EX-102, Addressing / Directories Working Group, Février 1988.

"Proposed New direction concerning EDI Use of X.500 Directory", J. Pilkington, British Telecom, Juin 1989.

Facultés Universitaires Notre-Dame de la Paix
Institut d'Informatique

Rue Grandgagnage, 21, B-5000 Namur (Belgium)

Introduction à X.500 : principes et définitions abstraites

Annexes

Promoteur : Professeur Philippe van BASTELAER

Mémoire présenté en vue de l'obtention
du titre de Licencié et Maître en Informatique

Année académique 1991-1992

Annexe X.500.1

SECTION 1
INTRODUCTION TO
NOTATION AND BASIC ENCODING

MHS238 087

MH130.026

X.409 STANDARD NOTATION

Recommendation X.409 defines the presentation transfer syntax and notation used by application layer protocols in message handling systems.

In the architecture of OSI, application entities exchange information in the form of application protocol data units (APDUs). The application protocol specification identifies the information contained in each APDU.

Each piece of information in an APDU is considered to have a type as well as a value. A **data type** (or **type** for short) is a class of information (for example, numeric or textual). A **data value** (or **value** for short) is an instance of such a class (for example, a particular number or fragment of text).

Recommendation X.409 defines several generally useful types and construction techniques from which APDUs are constructed in other Recommendations. This Recommendation presents and gives an example of the intended use, standard notation and standard representation of each type.

The **standard notation** of a type (or **notation** for short) is the language and conventions employed to denote either the type itself or a value of the type. The type notation is used to specify the structure and primitive elements of objects defined by application protocols. The value notation is used to give examples of such objects and to specify distinguished values of them. By providing a standard notation, application protocols can be specified without reference to how the APDUs are encoded for transmission.

As an example of standard notation, **INTEGER** is the standard notation used to denote an integer type, and either **17** or **11H** are the standard notations used to denote the integer value seventeen.

Note that information in APDUs is generally structured, i.e., composed of parts. For example, the information **Date** is composed of **Day-of-month**, **Month**, and **Year**. This means that the standard notation has to be able to capture the overall structure of the information as well as the primitive values of the pieces of information. Also note that different uses of the information and different values have to be distinguished: Day-of-month and Month could both be denoted as INTEGER, but they are in fact different pieces of information, and they allow different valid values, sometimes in sophisticated combinations. Recommendation X.409 provides mechanisms to handle all these requirements.

X.409 STANDARD NOTATION

- APPLICATION PROTOCOL DATA UNITS (APDUs) CONTAIN INFORMATION

  EXAMPLE: DATE = DAY-OF-MONTH | MONTH | YEAR

- X.409 STANDARD NOTATION =

  LANGUAGE FOR DENOTING WHAT INFORMATION IS CONTAINED IN EACH APDU

- EACH PIECE OF INFORMATION HAS A TYPE AND A VALUE

  TYPE = CLASS OF INFORMATION

  VALUE = INSTANCE OR MEMBER OF CLASS

MH130A 026

X.409 STANDARD NOTATION - EXAMPLE

- APDU: DATE = DAY-OF-MONTH | MONTH | YEAR

- EXAMPLE: PIECE OF INFORMATION: DAY-OF-MONTH
  - TYPE: INTEGER
  - VALUE: 1,2,...,31

- NOTES ON EXAMPLE:
  - LOTS OF DIFFERENT INFO COULD BE 'INTEGERS'
  - DIFFERENT MONTHS ALLOW DIFFERENT VALUES
  - SOME VALUES ARE DISTINGUISHED ('FIRST-DAY' ALWAYS HAS A VALUE 1)

MH130B 046

MH131.026

X.409 STANDARD REPRESENTATION

The **standard representation** of a type (or **representation** for short) is the set of rules for encoding values of that type for transmission as a sequence of octets. In the architecture of OSI, the Presentation Layer is responsible for encoding structured information into a sequence of octets. For this reason, the X.409 standard representation is known as the MHS **Presentation Transfer Syntax**. The X.409 standard representation of a value also encodes its type and length.

As an example of standard representation, the octet-string 020111 hexadecimal is the standard representation of the information **INTEGER 17**. This representation includes the type 02 hexadecimal (which is the standard encoding of INTEGER type), the length 01 hexadecimal (which is the standard encoding of a one-octet length), and the value 11 hexadecimal (which is the standard encoding of the integer value 17).

Note the advantage of the type-length-value style of encoding. With non-TLV encoding, the structured information may all be present, but you need a template to figure out where it is; without a template, structured information such as **Date** could be ambiguously encoded. With TLV encoding, the template itself is an integral part of the encoding scheme, and the encoding is unambiguously self-identified.

X.409 STANDARD REPRESENTATION

- APDUs ARE ENCODED INTO AGREED PRESENTATION TRANSFER SYNTAX
- X.409 STANDARD REPRESENTATION =
  ENCODING OF APDUs INTO OCTET-STRINGs
- ENCODING CONTAINS TYPE-LENGTH-VALUE (TLV) OF EACH PIECE OF INFORMATION

MH131A.046

X.409 STANDARD REPRESENTATION - EXAMPLE

- EXAMPLE:

T = D-O-M	T = MONTH	T = YEAR
L = 2	L = 2	L = 4
V = "10"	V = "03"	V = "1986"

NON-TLV: "10031986"
"19860310"? (ISO 2014)

- EXAMPLE: INTEGER 17:

T	L	V
02	01	11

(HEX)

X.409 CONCRETE SYNTAX

The term **syntax** refers to the **form** of information as distinguished from its **meaning**. For example, "INTEGER 17" is purely syntactic information; the syntax gives no indication of what the information means, i.e., what an "INTEGER" is, what "17" means, or how the information is to be used.

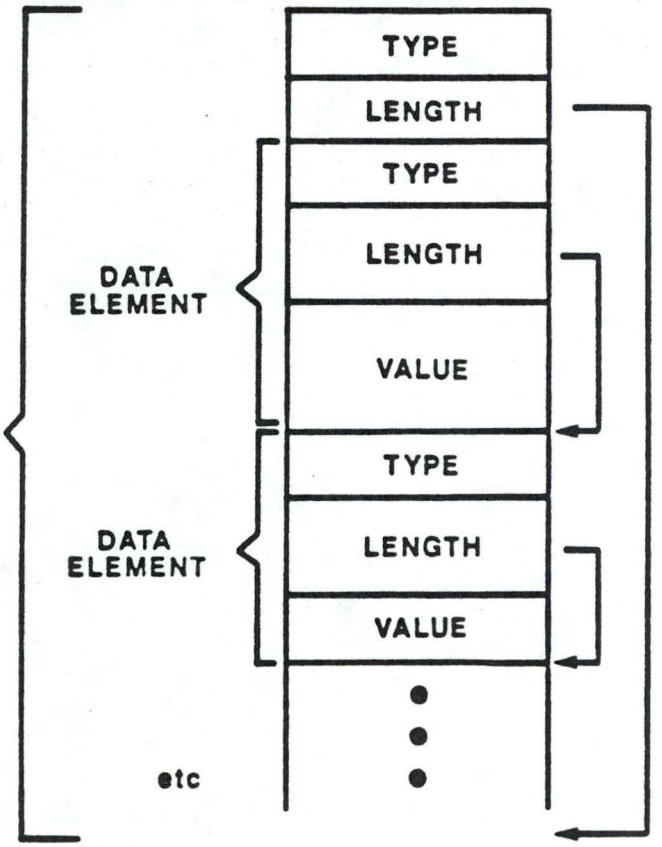
Syntactic information may appear either in a **notational** context (i.e., for an information specification purpose) or in an **encoding** context (i.e., for an information transmission purpose). These contexts are sometimes distinguished by referring to syntactic notation as **abstract syntax**, and to syntactic encoding as **concrete syntax**.

The X.409 concrete syntax -- the standard representation or encoding of MHS APDU's for transmission -- is a **type-length-value (TLV)** code. A TLV scheme encodes information as a sequence of **data elements** (or **elements** for short), each of which is either a **primitive** data element (i.e., is "atomic", has no parts) or is a **constructed** data element (i.e., is made up of other elements). Each element has its type, length, and value fully and explicitly encoded, together with an indication of whether it is primitive or constructed.

X.409 has adopted unique terminology for the three **components** of its data element encodings: **identifier-length-contents** instead of **type-length-value** -- but they mean the same thing as "TLV". The X.409 standard representation of a value of each type is an element having three components, which always appear in the following order. The **Identifier** distinguishes one type from another and governs the interpretation of the Contents. The **Length** specifies the length of the Contents. The **Contents** is the substance of the element, containing the primary information the element is intended to convey.

X.409/ASN.1 TRANSFER SYNTAX

IS A "TLV" CODE (TYPE-LENGTH-VALUE):
X.409 TERMINOLOGY = "IDENTIFIER-LENGTH-CONTENTS"



MH133.026

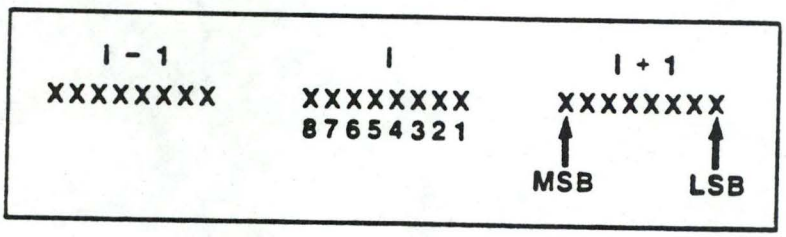
X.409 ENCODING - NUMBERING OF OCTETS AND BITS

An element or any of its components may be viewed as a sequence of an integral number, **n**, of octets numbered 0 through **n-1**. Each octet may be viewed in turn as a sequence of eight bits numbered 8 through 1.

Octet **i-1** is shown to the left of Octet **i** and Bit **j+1** to the left of Bit **j** on the printed page, and the adjectives "first" and "last" appeal to these particular octet and bit orderings.

The abbreviations "MSB" and "LSB" stand for "most significant bit" and "least significant bit", respectively.

X.409 ENCODING - NUMBERING OF OCTETS AND BITS



- EACH ELEMENT OR COMPONENT IS A SEQUENCE OF N OCTETS NUMBERED FROM 0 TO N-1
- EACH OCTET IS A SEQUENCE OF 8 BITS NUMBERED FROM 8 TO 1

MH133.026

PR031 026

X.409 ENCODING - IDENTIFIER

The Identifier distinguishes one type from another and governs interpretation of the Contents. It is one or more octets in length.

Four classes of types are distinguished by means of the Identifier: universal, application-wide, context-specific, and private-use. **Universal** types are generally useful, application-independent types. **Application-wide** types are more specialized, being peculiar to a particular application; they are defined in each specific other CCITT Recommendation or ISO Standard by means of the Tagged type (described later). **Context-specific** types are also defined using Tagged; however, they are used within an even more limited context -- for example, that of a Set -- and their Identifiers are assigned so as to be distinct only within that limited context. **Private-use** types are reserved for private use, outside of CCITT or ISO standards, and are to be discouraged in general because they limit the open interworking of systems.

The four type classes are distinguished by means of the first and second bits (i.e., Bits 8-7) of the first Identifier octet, as shown in the figure.

Two forms of data elements are distinguished: primitive and constructor. **A primitive** element is one whose Contents is atomic, i.e., has no further internal structure of data elements. **A constructor** element is one whose Contents is itself a data element or a series of data elements. Constructor elements are thus recursively defined, i.e., a constructor element may contain other constructor elements.

The two element forms are distinguished by means of the third bit (i.e., Bit 6) of the first Identifier octet, as shown in the figure.

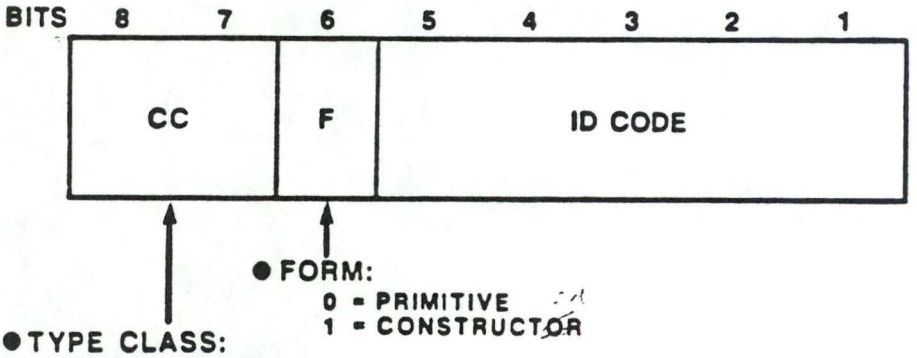
The remaining five bits (i.e., Bits 5-1) of the first Identifier octet encode a numeric ID Code that distinguishes one data type from another of the same class. These bits encode ID Codes in the range 0-30 as unsigned binary numbers whose MSB and LSB are Bit 5 and Bit 1, respectively. If the ID Code is greater than 30, Bits 5-1 have the value 11111 binary, and the ID Code is encoded in one or more extension octets, as shown in the figure.

Bit 8 of each extension octet indicates whether it is the last: Bit 8 of the last octet is zero; Bit 8 of each preceding extension octet is one. Bits 7-1 of the extension octets collectively encode ID Codes greater than 30. Conceptually, these groups of bits are concatenated to form an unsigned binary number whose MSB is Bit 7 of the first extension octet and whose LSB is Bit 1 of the last extension octet. The ID Code shall be encoded in the fewest possible octets, that is, with no leading octets with Bits 7-1 all zero.

The structure of single and multi-octet Identifiers are summarized in the figure, in which the bits denoted by "c" encode the class of the data type, the bit denoted by "f" encodes the form of the element, and the bits denoted by "i" encode the ID Code assigned to the type.

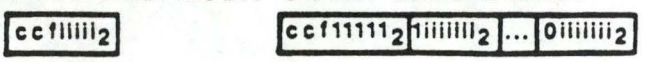
X.409 ENCODING - 1

"IDENTIFIER"



- FORM:
  - 0 = PRIMITIVE
  - 1 = CONSTRUCTOR
- TYPE CLASS:
  - 00 = UNIVERSAL
  - 01 = APPLICATION-WIDE
  - 10 = CONTEXT-SPECIFIC
  - 11 = PRIVATE USE

  00000000 RESERVED
- ID CODE:
  - 00000-11110 = ASSIGNED ID CODES
  - 11111 = EXTENSION
- SINGLE AND MULTI-OCTET IDENTIFIERS:



PRO31.097

PR032.026

X.409 ENCODING - LENGTH AND CONTENTS

The Length specifies the length L in octets of the Contents and is itself variable in length.

The Length may take any of three forms: short, long, and indefinite.

The **short** form is one octet long and shall be used in preference to the long form when L is less than 128. Bit 8 has the value zero, and Bits 7-1 encode L as an unsigned binary number whose MSB and LSB are Bit 7 and Bit 1, respectively. For example, L = 38 is encoded as: 00100110 binary.

The **long** form is from 2 to 127 octets long and is used when L is greater than or equal to 128 and less than 2**1008 (1008 = 8*126), except where the indefinite form is used. Bit 8 of the first octet has the value one. Bits 7-1 of the first octet encode a number one less than the size of the Length in octets as an unsigned binary number whose MSB and LSB are Bit 7 and Bit 1, respectively. L itself is encoded as an unsigned binary number whose MSB and LSB are Bit 8 of the second octet and Bit 1 of the last octet, respectively. This binary number shall be encoded in the fewest possible octets, with no leading octets having the value zero.

The **indefinite** form is one octet long and may (but need not) be used in place of the short or long form whenever the element is a constructor. It has the value 10000000 binary. When this form is employed, a special **end-of-contents (EOC)** element terminates the Contents. There is no notation for EOC, because although considered part of the Contents syntactically it has no semantic significance and is not mentioned in descriptions of particular types. The representation for EOC is 0000 hexadecimal (i.e., universal class 0, form primitive, length of Contents zero).

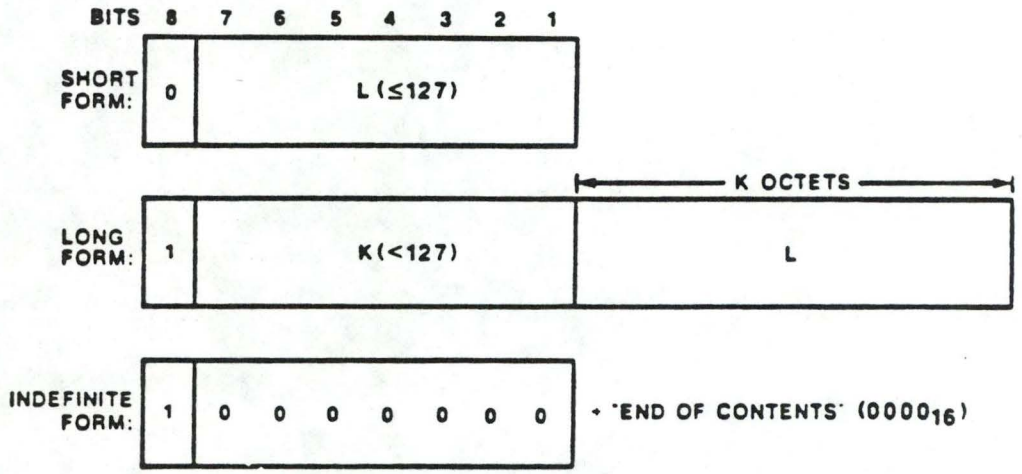
Note that no valid form of Length has the first (or only) octet equal to 11111111 binary. This value is reserved for possible future extension.

The Contents is the substance of the element and contains the primary information the element is intended to convey. It is variable in length, but always a multiple of eight bits, and is interpreted in a type-dependent way (i.e., depending on the value of Identifier). If the element is a constructor, the Contents itself comprises zero or more elements; elements are thus recursively defined.

In some cases, the Contents comprises an ordered series of entities -- the bits of a bit string, the octets of an octet string, or the elements of a sequence. In these cases, the order in which entities appear in the representation corresponds to the order in which they appear in the notation. The entity occupying the leftmost position on the printed page appears first in the Contents. The entity occupying the rightmost position appears last.

X.409 ENCODING – 2

'LENGTH'



'CONTENTS' =
ASSIGNED VALUE OF IDENTIFIED TYPE
(CONTAINS PRIMARY INFORMATION WHICH
THE DATA ELEMENT IS INTENDED TO CONVEY)

PR032.056

MH136.026

TYPES DEFINED/ALLOWED IN X.409

X.409 contains a specification of ten built-in types, including primitive types such as Boolean and Integer, and constructor types such as Sequence and Tagged.

X.409 also contains seven defined types such as IA5 String and Numeric String. These are specified using the standard notation and the built-in types.

In addition, X.409 specifies ways to define user-defined types, macros, and modules. User-defined types such as Day-of-Month and Primary-Color are defined from built-in types and other defined types using the standard notation. The macro facility allows non standard type and/or value notation to be specified for a particular new data type, such as OPERATION and ERROR as defined and used in X.410. The module facility allows definitions to be grouped into modular packages for convenience in handling large collections of definitions, such as P1.ORName and P2.UAPDU as used in structuring MHS.

TYPES DEFINED/ALLOWED IN X.409

- PRIMITIVE - ATOMIC BUILT-IN TYPES
  - Examples: Boolean, Integer
- CONSTRUCTOR - BUILT-IN BUILDING TOOLS
  - Examples: Sequence, Tagged
- DEFINED - X.409 CONSTRUCTED TYPES
  - Examples: IA5 String, Numeric String
- USER-DEFINED TYPES - TYPES DEFINED BY X.409 USERS USING STANDARD NOTATION
  - Examples: Day-of-Month, Primary-Color
- MACROS - ALLOW X.409 USERS TO DEFINE NON-STANDARD NOTATION
  - Examples: OPERATION, ERROR (X.410)
- MODULES - ALLOW X.409 USER TO GROUP DEFINITIONS INTO MODULAR PACKAGES
  - Examples: P1.ORName, P2.UAPDU

MH136.026

BNF PRIMER

Before beginning the detailed definition of the MHS data types, we give a brief introduction to BNF.

Backus-Naur Form (BNF) is the most well-known formal method for defining computer languages.

A computer language or **formal language** is a set of strings of symbols. For any string, it must be possible to determine uniquely whether the string is in the language or not in the language. BNF is a method for **producing** (i.e., constructing) all the strings in the language being defined.

A **terminal** is a symbol that appears literally in the language being defined. For example, for the BINARYNUMBERS language, there are two terminals: **0** and **1**.

A **nonterminal** is a symbol that is a placeholder. It is defined to be equivalent to either a particular series of symbols -- either terminal or nonterminal -- or any of several such series. For example, for the **BINARYNUMBERS language**, **BinaryDigit** is a placeholder that is defined to be equivalent to either of the terminals **0** or **1**. Whenever you see the nonterminal **BinaryDigit**, you can replace it with either a **0** or a **1** and you'll have a valid string (series of symbols) in the language.

The **equivalence** operator, denoted by the peculiar configuration of symbols **::=** (which may be read "is" or "is defined as" or "produces"), assigns a value to a nonterminal. The nonterminal appears on the left hand side, and the alternative values that may be assigned appear on the right hand side, separated by **alternative** operators, denoted by vertical bars. Such an assignment is called a **production**.

For example, referring to the figure, the BINARYNUMBERS language is defined by two productions:

A **BinaryNumber** is defined as either a **BinaryDigit** or a **BinaryDigit** followed by a **BinaryNumber**, where a **BinaryDigit** is defined as either a **0** or a **1**. This definition will generate any and all binary numbers and only binary numbers, as you will find out if you try it.

BNF PRIMER

- BACKUS-NAUR FORM (BNF) IS A FORMAL METHOD FOR WRITING COMPUTER LANGUAGES
- LANGUAGE = SET OF STRINGS OF SYMBOLS
- FOUR MAIN CONCEPTS OF BNF:
  - TERMINAL = LITERAL OF LANGUAGE BEING DEFINED
  - NONTERMINAL = PLACE-HOLDER
  - EQUIVALENCE OPERATOR ("::=")
  - ALTERNATIVE OPERATOR ("|")

MH137A.046

BNF PRIMER (CONT.)

- Example: THE "BINARY NUMBERS" LANGUAGE CAN BE DEFINED BY 2 BNF PRODUCTIONS :
  - BinaryNumber ::= BinaryDigit | BinaryDigit BinaryNumber
  - BinaryDigit ::= 0|1
- APPLYING THIS DEFINITION PRODUCES THE LANGUAGE
  - {0, 1, 00, 01, 10, 11, 000,...}

MH138.026

X.409 STANDARD NOTATION CONVENTIONS

To help in reading the X.409 language, a number of conventions have been adopted.

Nonterminals are rendered in **bold**, and generally begin with a capital letter except for the four special cases identified in the figure, and except for value definitions (see below).

Comments may be embedded anywhere in the notation. They are preceded by two hyphens ("--") and terminated by either two hyphens or the end of a line (see 1A5).

X.409 STANDARD NOTATION CONVENTIONS

- NONTERMINALS ARE RENDERED IN BOLD
- NONTERMINALS DEFINED IN X.409 BEGIN WITH CAPITAL LETTERS, EXCEPT FOR THESE 4:
  - string = SEQUENCE OF ZERO OR MORE CHARS
  - identifier = SEQUENCE OF ONE OR MORE CHARS: CAPITALS/SMALLS/NUMBERS/HYPHEN, FIRST IS A LETTER
  - number = NONNEGATIVE INTEGER: DECIMAL OR HEXADECIMAL ("H")
  - empty = STRING OF ZERO SYMBOLS
- EMBEDDED COMMENTS:
  - PRECEDED BY TWO HYPHENS ("--")
  - TERMINATED BY "--" OR END-OF-LINE

MM138.026

MH139.026

X.409 DEFINITIONS - NOTATION: TYPE, VALUE, MACRO

Defined types are fabricated from the X.409 **built-in** types and other defined types using the notation shown in the figure. Defining a type assigns a reference name to it (its **type name**), by which it shall be uniquely known throughout the scope of its definition (which varies according to its **class** -- see below). By convention, each type name begins with a capital letter. Wherever a type specification is called for by the standard notation, the type name of a defined type may be supplied in its place.

Values are defined using the notation shown in the figure. Defining a value assigns a reference name to it (its **value name**). By convention, each value name begins with a small letter. Wherever a value specification is called for by the standard notation, the value name of a defined value may be supplied in its place. Similarly, in most places where a number is required, the value name of a defined Integer value may be supplied in its place. This flexibility of using names of values instead of the values themselves is very natural and will cause no confusion when reading X.409 statements.

For example, we can define a new type **PrimaryColor** as an INTEGER with distinguished values **red(0)**, **yellow(1)**, and **blue(2)**. Then we don't have to remember what the integer values are, but can just use the names of the values instead: **yellow** instead of 1. Furthermore, we can define new value names; for example, we can assign a value name **defaultColor** of **PrimaryColor** is **yellow**, and then any time we use **defaultColor** in an expression, it will always mean **yellow**, which in turn is defined as the integer 1. The language notation takes care of "remembering" the assignments, so we as humans can just use the natural notational expressions. This makes X.409 definitions remarkably easy to read once you get the hang of it.

It is occasionally useful to define **non** standard type and/or value notation for a particular type. Such notation is defined by means of **a macro**, using the notation shown in the figure. Defining a macro assigns a reference name to it (its **macro name**). The macro name is always the first symbol in the type notation defined by the macro (**identifier** in the figure); by convention, each macro name is all capital letters. The **MacroBody** specifies the desired non-standard notation using BNF. The macro notation is specified in detail in X.409.

X.409 DEFINITIONS - NOTATION

TypeDefinition :: = identifier ":: =" Type

Example:

PrimaryColor :: = INTEGER {red(0), yellow(1), blue(2)}

ValueDefinition :: = identifier Type ":: =" Value

Example:

defaultColor PrimaryColor :: = yellow

MacroDefinition :: = identifier MACRO ":: =" BEGIN MacroBody END

MH139.026

MH140.026

X.409 DEFINITIONS - NOTATION: MODULE

It is often convenient to group related type, value, and macro definitions, for example, those of a particular protocol specification. Collections -- called **modules** -- of zero or more type, value, and macro definitions are defined using the notation shown in the figure. Defining a module assigns a reference name to it (its **module name**). By convention, each module name begins with a capital letter. The reference names of the types, values, and macros defined within a module must be distinct.

Within a module, wherever a type or value specification is called for by the standard notation, the reference name of a type, value, or macro defined in another or the same module, or the reference name of any of the universal types defined in X.409, may be supplied in its place. If the type, value, or macro is defined in another module, its reference name is preceded by that of the module in which the type, value, or macro is defined; the two reference names are separated by a period (".").

In the example shown in the figure, we group definitions of the type **PrimaryColor** and the value **defaultColor** into a module named **Color**. From outside the module, these definitions would then be referenced by the names **Color.PrimaryColor** and **Color.defaultColor**, respectively.

In the following pages, we specify the ten built-in types defined in X.409.

X.409 DEFINITIONS - NOTATION (CONT.)

```
ModuleDefinition    :: =  identifier DEFINITIONS ":: =" BEGIN ModuleBody END

ModuleBody          :: =  DefinitionList | empty

DefinitionList      :: =  Definition | DefinitionList Definition
Definition          :: =  TypeDefinition | ValueDefinition | MacroDefinition
```

Example:

```
Color DEFINITIONS :: =
BEGIN
PrimaryColor :: = INTEGER {red(0), yellow(1), blue(2)}
defaultColor PrimaryColor :: = yellow
END
```

MH140.026

NOTES

SECTION 2
FUNDAMENTAL TYPES

MM268.097

MH141.026

X.409 BOOLEAN TYPE

A Boolean represents a logical quantity that can assume either of two values, true or false.

The notation for the Boolean type is the keyword **BOOLEAN**. The notation for a Boolean value is either of two keywords, **TRUE** or **FALSE**.

The representation for a Boolean is an element whose class is universal, whose form is primitive, whose ID Code is 1, and whose Contents is a single octet that encodes the value of the Boolean. **FALSE** is encoded as all bits zero, **TRUE** as any other combination of bits.

X.409 BOOLEAN TYPE

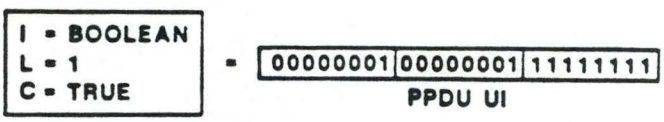
BooleanType :: = BOOLEAN

BooleanValue :: = TRUE | FALSE

MH141A.026

X.409 BOOLEAN TYPE - EXAMPLE

● Example: ILearnedSomethingToday :: = BOOLEAN



MH141B.026

X.409 INTEGER TYPE

An Integer represents an integer; an integer value may be positive, zero, or negative.

The notation for an Integer type is the keyword **INTEGER**, optionally followed by distinguished values and the distinct reference names assigned to them. By convention, the value names begin with small letters. The notation for an Integer value is a signed number or one of the assigned value names.

The representation for an Integer is an element whose class is universal, whose form is primitive, whose ID Code is 2, and whose Contents is the value of the Integer, encoded as a twos complement binary number whose MSB and LSB are bit 8 of the first octet and bit 1 of the last octet, respectively. The value of the integer shall be encoded in the fewest possible octets.

X.409 INTEGER TYPE

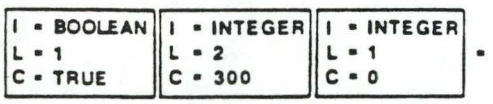
```
IntegerType       ::= INTEGER | INTEGER {NamedNumberList}
IntegerValue      ::= number | - number | identifier

NamedNumberList   ::= NamedNumber | NamedNumberList, NamedNumber
NamedNumber       ::= identifier (number)
```

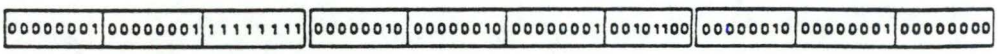
MH142A.026

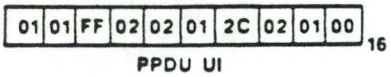
X.409 INTEGER TYPE – EXAMPLE

- Example: HowMuchItCost ::= INTEGER

  Where ::= INTEGER { asn(0), osi(1), bar(2) }







MH142B.097

MH143.026

X.409 BIT STRING TYPE

A Bit String represents an ordered set of zero or more bits.

The notation for a Bit String type is the keywords **BIT STRING**, optionally followed by the numbers of distinguished bits and the distinct reference names assigned to them. By convention, the reference names begin with small letters. The first bit is numbered zero. The notation for a Bit String value is a series of binary or hexadecimal digits enclosed in apostrophes and followed by the capital letter "B" or "H", respectively; or the assigned reference names of the one bits. The highest numbered bit that is explicitly specified is the value's last bit. Where such a convention is appropriate, missing bits are assumed to be zero.

The representation for a Bit String is an element whose class is universal, whose form is either primitive or constructor, whose ID Code is 3, and whose Contents depends upon the form. If the form is primitive, the Contents is the bits of the Bit String, packed eight to an octet and preceded by a single octet that encodes the number of unused bits in the final octet of the Contents -- from zero to seven -- as an unsigned binary number whose MSB and LSB are bit 8 and bit 1, respectively. If the form is constructor, the Contents is an ordered set of zero or more Bit Strings in their standard representations -- as if Bit String were defined as **[UNIVERSAL 3] IMPLICIT SEQUENCE OF BIT STRING**. Each component Bit String represents a **segment** -- zero or more bits -- of the overall Bit String. The number of bits in all segments but the last is a multiple of eight. Segment boundaries are not significant, that is, they carry no information. The constructor form is typically used in conjunction with the indefinite form of Length for very long Bit Strings whose total length may not be readily available.

Note that a Bit String is ordered in the octets that form its representation such that bit 0 of the Bit String corresponds to bit 8 of the first octet, bit 7 to bit 1 of the first octet, bit 8 to bit 8 of the second octet, etc.

X.409 BIT STRING TYPE

```
BitStringType      ::= BIT STRING | BIT STRING {NamedNumberList}
BitStringValue     ::= 'string' B | 'string' H [{IdentifierList}]

NamedNumberList    ::= NamedNumber | NamedNumberList , NamedNumber
NamedNumber        ::= identifier (number)

IdentifierList     ::= identifier | IdentifierList, identifier
```

MH143A 026

X.409 BIT STRING TYPE - EXAMPLE

- Example:

```
PersonalStatus ::= BIT STRING {married(0), employee(1),
                               veteran(2), collegeGraduate(3), female(4)}

value: '101'B = 'A'H = {married, veteran}
```

encoding:

00000011	00000010	00000011	10100000

= | 03 | 02 | 03 | A0 |$_{16}$

MH143B.097

MH144.026

X.409 OCTET STRING TYPE

An Octet String represents an ordered set of zero or more octets.

The notation for an Octet String type is the keywords **OCTET STRING**. The notation for an Octet String value takes either of two forms. The first form is a series of binary or hexadecimal digits enclosed in apostrophes and followed by the capital letter "B" or "H", respectively. The highest numbered octet that is explicitly specified at least partially is the value's last octet. Unspecified bits, if any, in the last octet are assumed to be zero. The second form is a series of characters enclosed in quotation marks. A quotation mark may appear in the series but must be doubled to distinguish it from the quotation mark that terminates the series. The interpretation of this second form is context-specific. Every use of it must be accompanied by a detailed specification of the characters that are allowed, their graphical depictions, and their representations as sequences of octets.

The representation for an Octet String is an element whose class is universal, whose form is either primitive or constructor, whose ID Code is 4, and whose Contents depends upon the form. If the form is primitive, the Contents is the octets of the Octet String. If the form is constructor, the Contents is an ordered set of zero or more Octet Strings in their standard representations -- as if Octet String were defined as **[UNIVERSAL 4] IMPLICIT SEQUENCE OF OCTET STRING**. Each component Octet String represents a segment -- zero or more octets -- of the overall Octet String. Segment boundaries are not significant, that is, they carry no information. The constructor form is typically used in conjunction with the indefinite form of Length for very long Octet Strings whose total length may not be readily available.

Note that the octets in an Octet String are numbered from 0 to n. An Octet String is ordered in the octets that form its representation such that octet 0 of the Octet String corresponds to the first octet and octet n to the last octet of the representation.

X.409 OCTET STRING TYPE

OctetStringType :: = OCTET STRING

OctetStringValue :: = 'string' B | 'string' H | "string"

MH144.026

MH145.026

X.409 NULL TYPE

A Null represents a valueless placeholder.

The notation for the Null type and the Null value is the keyword **NULL**.

The representation for a Null type is an element whose class is universal, whose form is primitive, whose ID Code is 5, and whose Contents is unused and absent.

X.409 NULL TYPE

```
NullType  ::= NULL

NullValue ::= NULL
```

MH145.026

MH146 026

X.409 SEQUENCE TYPE

A Sequence represents an ordered set of zero or more values called its **elements**.

The notation for a Sequence type is the keyword **SEQUENCE**, followed by any constraints imposed upon the elements. Three forms of Sequence are allowed with different constraints.

A Sequence may have elements that are variable in number but of one type, in which case that type is specified.

Alternatively a Sequence may have elements that are fixed in number, and possibly of several (although not necessarily distinct) types, in which case those types and the distinct reference names optionally assigned to the elements are specified.

A Sequence of multiple types may have elements that are optional, that is, may be omitted at the discretion of the entity constructing the Sequence. In this case all elements of the Sequence, including optional elements, must be of distinct types. Default values may (but need not) be associated with optional elements. Optional elements are identified by means of the keyword **OPTIONAL**, if no default value is specified, or **DEFAULT**, otherwise.

The notation for a Sequence value is the reference names (if any) and values of its elements. Note that in case of a Sequence of multiple types with optional elements, failure to assign a reference name to each member may render the value notation ambiguous: it may not be clear which member is being specified.

A construct **COMPONENTS OF** is defined in X.409 to include the elements of another Sequence among those of the Sequence being defined.

The representation for a Sequence is an element whose class is universal, whose form is constructor, whose ID Code is 16, and whose Contents is the elements of the Sequence, ordered and in their standard representations.

X.409 SEQUENCE TYPE

```
SequenceType     ::= SEQUENCE | SEQUENCE OF Type | SEQUENCE {ElementTypes}

SequenceValue    ::= {ElementValues}

ElementTypes     ::= OptionalTypeList | empty
OptionalTypeList ::= OptionalType | OptionalTypeList, OptionalType
OptionalType     ::= NamedType | NamedType OPTIONAL | NamedType DEFAULT Value
NamedType        ::= identifier Type | Type

ElementValues    ::= NamedValueList | empty
NamedValueList   ::= NamedValue | NamedValueList, NamedValue
NamedValue       ::= identifier Value | Value
```

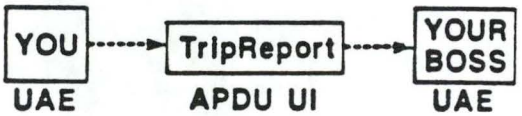
MH146A.026

X.409 SEQUENCE TYPE - EXAMPLE

- Example:

TripReport ::= SEQUENCE OF DailyReport

-- must be ordered chronologically,
-- earliest date first



MH146B.026

MH147 026

X.409 SET TYPE

A Set represents an unordered set of zero or more values called its **members**.

The notation for a Set type is the keyword **SET**, followed by any constraints imposed upon the members. Three forms of Set are allowed, with different constraints.

A Set may have members that are variable in number but of one type, in which case that type is specified.

Alternatively a Set may have members that are fixed in number and of distinct types, in which case those types and the distinct reference names optionally assigned to the members are specified.

A Set of multiple types may have members that are optional, that is, may be omitted at the discretion of the entity constructing the Set. In this case as well, all members of the Set, including optional members, must be of distinct types. Default values may (but need not) be associated with optional members. Optional members are identified by means of the keyword **OPTIONAL**, if no default value is specified, or **DEFAULT**, otherwise.

The notation for a Set value is the reference names (if any) and values of its members. Note that in case of a Set of multiple types, failure to assign a reference name to each member may render the value notation ambiguous: it may not be clear which member is being specified.

A construct **COMPONENTS OF** is defined in X.409 to include the elements of another Set among those of the Set being defined.

The representation for a Set is an element whose class is universal, whose form is constructor, whose ID Code is 17, and whose Contents is the specified members of the Set, in any order but in their standard representations. The elements that represent the members must have distinct Identifiers. This requirement is commonly met by context-specifically tagging each Set member using the Tagged type (described next).

X.409 SET TYPE

```
SetType          ::= SET | SET OF Type | SET {MemberTypes
SetValue         ::= {MemberValues

MemberTypes      ::= OptionalTypeList | empty
OptionalTypeList ::= OptionalType | OptionalTypeList, OptionalType
OptionalType     ::= NamedType | NamedType OPTIONAL | NamedType DEFAULT Value
NamedType        ::= identifier Type | Type

MemberValues     ::= NamedValueList | empty
NamedValueList   ::= NamedValue | NamedValueList, NamedValue
NamedValue       ::= identifier Value | Value
```

MH147A.026

X.409 SET TYPE - EXAMPLE

- Example:

```
Date ::= SET {day [0] DayOfMonth,
              month [1] MonthOfYear,
              year [2] IMPLICIT INTEGER OPTIONAL}

DayOfMonth ::= INTEGER {first(1)}

MonthOfYear ::= INTEGER {january(1), february(2),...(etc)}

value: {month march, day 10} = {day 10, month 3}
```


MH147B.007

MH148 026

X.409 TAGGED TYPE

A Tagged represents a value that has been tagged for identification. The Tagged type provides a means for creating new types from types that already exist, and in such a way that the new types are distinguishable from the old. A type thus defined can be universal, application-wide, context-specific, or private-use. The type of the value being tagged can be either explicit in the representation or implicit, except that a value of type Choice or Any may not be implicitly tagged.

Note that every value is tagged in the sense that its type can be determined from the Identifier component of the element that represents it. An Integer is inherently distinguishable from an IA5 String, for example. The Tagged type allows a value to be tagged at a higher level to reflect its semantics and/or its syntactical constraints (for example, the allowed range of an Integer or the characters allowed in an IA5 String). Thus, for example, a user name and country name can be distinguished even if both are (tagged) IA5 Strings.

The notation for a Tagged type is a numeric tag followed by the type of the value being identified. The tag must be distinct from the tags -- and, if the new type is universal, the ID Codes -- assigned to other types of the same class. If the new type is universal, application-wide, or private-use, the keyword **UNIVERSAL**, **APPLICATION**, or **PRIVATE**, respectively, precedes the tag. Otherwise, the type is context-specific. If the type of the value being tagged is implicit, the keyword **IMPLICIT** precedes the type. Otherwise, the type is explicit. The notation for a Tagged value is that of the value being identified. Note that the normal context for a context-specifically tagged data element is a Sequence, Set, or Choice.

The representation for a Tagged is an element whose class is that specified and whose ID Code is the tag. In the explicit case, the form of the element is constructor, and its Contents is the value being tagged, in its standard representation. In the implicit case, the form of the element is that of the value being tagged, and its Contents is the Contents (only) of the value being tagged. Implicitly tagging a value thus effectively alters its class and ID Code.

X.409 TAGGED TYPE

TaggedType :: = **Tag IMPLICIT Type | Tag Type**
TaggedValue :: = **Value**

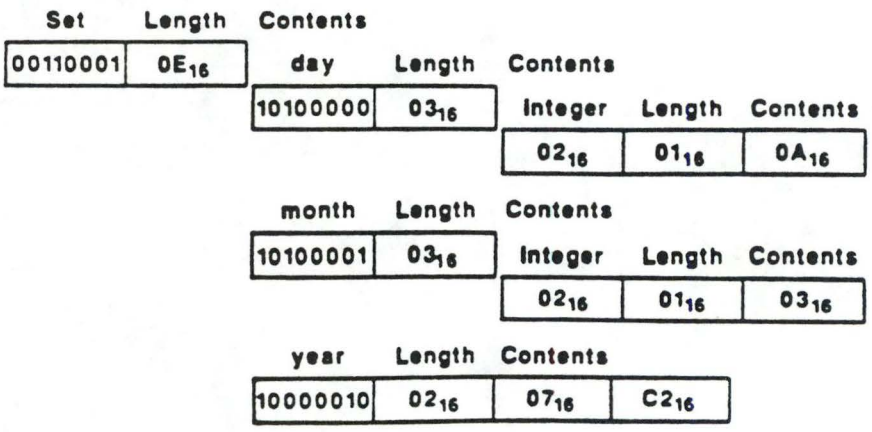
Tag :: = **[Class number]**
Class :: = **UNIVERSAL | APPLICATION | PRIVATE | empty**

MH1488.026

X.409 TAGGED TYPE - EXAMPLE

- **Example:**

type: Date
value: (day 10, month march, year 1986)



MH149 026

X.409 CHOICE TYPE

A Choice represents a value whose type is chosen from a set of one or more distinct alternatives.

The notation for a Choice type is the keyword **CHOICE**, followed by the types of the alternatives and the distinct reference names optionally assigned to them. By convention, the reference names begin with small letters. The notation for a Choice value is the reference name (if any) and value of the chosen alternative. Note that failure to assign a reference name to each alternative may render the value notation ambiguous: it may not be clear which alternative is being specified.

X.409 also defines a construct called a **bound Choice**, which represents a particular alternative selected from a previously defined unbound Choice type.

The representation for a Choice is that of the chosen alternative. The elements that represent the alternatives must have distinct identifiers. This requirement is commonly met by context-specifically tagging each Choice alternative using the Tagged type.

X.409 CHOICE TYPE

```
ChoiceType          :: = CHOICE {AlternativeTypeList}

ChoiceValue         :: = identifier Value | Value

AlternativeTypeList :: = NamedType | AlternativeTypeList, NamedType

NamedType           :: = identifier Type | Type
```

MH149A 026

X.409 CHOICE TYPE - EXAMPLE

- Example:
```
DailyReport ::= SEQUENCE {
    Date,
    CHOICE{
        [0] IMPLICIT MarketingReport,
        [1] IMPLICIT EducationReport},
    ANY OPTIONAL}
```

MH149B 026

MH150 026

X.409 ANY TYPE

An Any represents a value whose type is unrestricted, that is, chosen from the set of all built-in and defined types.

The notation for the Any type is the keyword ANY. The notation for an Any value is the chosen type followed by a value of that type.

The representation for an Any is that of the chosen type.

X.409 ANY TYPE

AnyType :: = ANY

AnyValue :: = Type Value

MH150.026

MH151.026

X.409 DEFINED TYPES

X.409 defines seven defined types. A defined type is one specified using the standard notation for type definitions. The formal definitions of the seven X.409 defined types are given in the figure.

An **IA5String** represents an ordered set of zero or more characters chosen from the International Reference Version of International Alphabet No. 5 (i.e, ASCII). The characters allowed and their graphical depictions and seven-bit numeric codes are those specified for IA5-IRV in Recommendation T.50. Each octet contains a single code. Bit 8 of each octet is zero, and Bits 7-1 correspond to b7-b1 of the code (using the T.50 bit numbering convention).

A **NumericString** represents an ordered set of zero or more characters that collectively encode numeric information in textual form. It models data entered from such devices as telephone handsets.

A **PrintableString** represents an ordered set of zero or more characters chosen from a subset of the printable characters. It models data entered from devices with a limited character repertoire (for example, Telex terminals).

A **T.61String** represents an ordered set of zero or more characters and presentation commands chosen from the set defined by Recommendation T.61. It models textual data suitable for processing by Teletex terminals. The characters and commands and their graphical depictions and eight-bit numeric codes are defined in T.61. Each octet contains a single code. Diacritically marked characters are represented by a pair of codes.

A **VideotexString** represents an ordered set of zero or more alphabetic characters, pictorial characters, pictorial drawing commands, display attribute commands, etc., chosen from the set defined by the Data Syntaxes of Recommendation T.101 or the options from Recommendation T.100. It models textual and graphical data suitable for processing by Videotex terminals. The characters and commands and their graphical depictions and seven-bit numeric codes are defined in T.100 and T.101. Each octet contains a single numeric code. Bit 8 of each octet is zero, and Bits 7-1 correspond to b7-b1 of the code.

A **GeneralizedTime** represents a calendar date and time of day to various precisions, as provided for by ISO 2014, ISO 3307, and ISO 4031. The time of day can be specified as local time only, UTC time only, or as both local and UTC time.

The **UTCTime** type is a particular form of GeneralizedTime especially suited for the MHS application. It is a numeric string of either ten (YYMMDDhhmm) or twelve (YYMMDDhhmmss) digits, followed by either the letter "Z" or an offset of the form "+hhmm" or "-hhmm" representing the difference from local time to UTC. (YY is the two low-order digits of the Christian era year and MM, DD, hh, mm, and ss are the month, day, hour, minute, and second, respectively.)

X.409 DEFINED TYPES

- IA5String ::= [UNIVERSAL 22] IMPLICIT OCTET STRING
  -- uses 7-bit IRV-IA5 (ASCII)
- NumericString ::= [UNIVERSAL 18] IMPLICIT IA5String
  -- characters allowed are digits 0,1,...,9 and space
- PrintableString ::= [UNIVERSAL 19] IMPLICIT IA5String
  -- alphanumerics, punctuation, +, =
- T.61String ::= [UNIVERSAL 20] IMPLICIT OCTET STRING
  -- text data suitable for Teletex terminals
- VideotexString ::= [UNIVERSAL 21] IMPLICIT OCTET STRING
  -- text and graphical data defined by T.100, T.101
- GeneralizedTime ::= [UNIVERSAL 24] IMPLICIT IA5String
  -- date and time of day to various precisions
- UTCTime ::= [UNIVERSAL 23] IMPLICIT GeneralizedTime
  -- date and UTC time to one minute or one second

MH151.026

X.409 WORKED-OUT EXAMPLE - DEFINITIONS

```
TripReport ::= SEQUENCE OF DailyReport
   -- must be ordered chronologically,
   -- earliest date first

DailyReport ::= SEQUENCE {
        Date,
        CHOICE{
           [0] IMPLICIT MarketingReport,
           [1] IMPLICIT EducationReport},
        ANY OPTIONAL}

Date ::= SET {day [0] DayOfMonth,
              month [1] MonthOfYear,
              year [2] IMPLICIT INTEGER OPTIONAL}

DayOfMonth ::= INTEGER {first(1)}
MonthOfYear ::= INTEGER {january(1),...(etc)}
EducationReport ::= SEQUENCE {
      ILearnedSomethingToday, HowMuchItCost, Where}

ILearnedSomethingToday ::= BOOLEAN
HowMuchItCost ::= INTEGER -- in dollars US
Where ::= INTEGER {asn(0), osi(1), bar(2)}
```

MH212.026

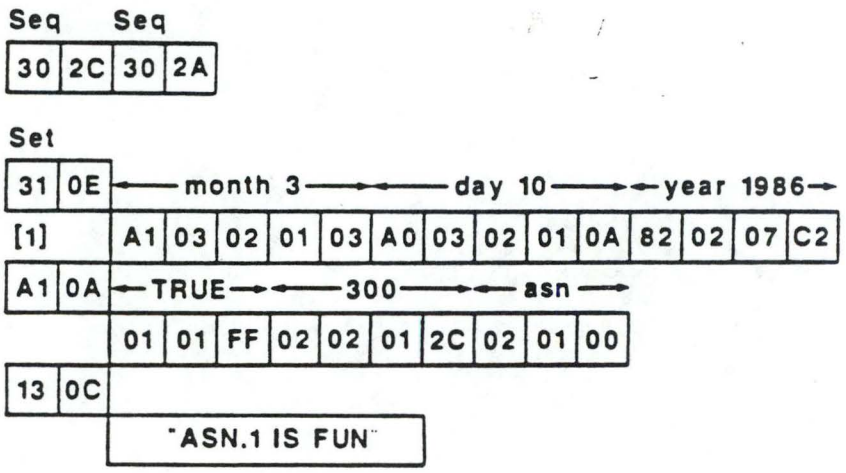
X.409 WORKED-OUT EXAMPLE - ENCODING

- **Example:**

type: TripReport

value: {{{month march, day 10, year 1986}
{TRUE, 300, asn},
PrintableString "ASN.1 IS FUN"}}

encoding:



MH213.097

Annexe X.500.2

ANNEX C - INFORMATION FRAMEWORK IN ASN.1

This Annex is part of the standard.

This Annex provides a summary of all of the ASN.1 type, value, and macro definitions contained in this Recommendation The definitions form the ASN.1 module **InformationFramework.**

```
InformationFramework {joint-iso-ccitt ds(5) modules(1) informationFramework(1)}
DEFINITIONS ::=
BEGIN

EXPORTS
     Attribute, AttributeType, AttributeValue, AttributeValueAssertion,
     DistinguishedName, Name, RelativeDistinguishedName,
     OBJECT-CLASS, ATTRIBUTE, ATTRIBUTE-SET, ATTRIBUTE-SYNTAX,
     Top, Alias,
     ObjectClass, AliasedObjectName,
     ObjectIdentifierSyntax, DistinguishedNameSyntax;

IMPORTS
     selectedAttributeTypes, selectedObjectClasses
          FROM   UsefulDefinitions {joint-iso-ccitt ds(5) modules(1) usefulDefinitions(0)}
     top
          FROM   SelectedObjectClasses selectedObjectClasses
     objectIdentifierSyntax, distinguishedNameSyntax, objectClass, aliasedObjectName
          FROM   SelectedAttributeTypes selectedAttributeTypes;

-- attribute data types --
Attribute                    ::=  SEQUENCE {
                                       type          AttributeType,
                                       values        SET OF AttributeValue
                                       -- at least one value is required --}

AttributeType                ::=  OBJECT IDENTIFIER

AttributeValue               ::=  ANY

AttributeValueAssertion      ::=  SEQUENCE {AttributeType, AttributeValue}

-- naming data types --
Name                   ::=   CHOICE {    -- only one possibility for now --
                                         RDNSequence }

RDNSequence                  ::=  SEQUENCE OF RelativeDistinguishedName

DistinguishedName            ::=  RDNSequence

RelativeDistinguishedName::=  SET OF   AttributeValueAssertion
```

```
-- macros --
OBJECT-CLASS MACRO ::=
BEGIN

    TYPENOTATION            ::=   SubclassOf
                                  MandatoryAttributes
                                  OptionalAttributes

    VALUENOTATION           ::=   value (VALUE OBJECT IDENTIFIER)

    SubclassOf              ::=   "SUBCLASS OF" Subclasses | empty
    Subclasses              ::=   Subclass | Subclass "," Subclasses
    Subclass                ::=   value(OBJECT-CLASS)

    MandatoryAttributes     ::=   "MUST CONTAIN {"Attributes"}" | empty
    OptionalAttributes      ::=   "MAY CONTAIN {"Attributes"}" | empty

    Attributes              ::=   AttributeTerm | AttributeTerm "," Attributes
    AttributeTerm           ::=   Attribute | AttributeSet
    Attribute               ::=   value (ATTRIBUTE)
    AttributeSet            ::=   value (ATTRIBUTE-SET)

END

ATTRIBUTE-SET MACRO ::=
BEGIN

TYPE NOTATION       ::=   "CONTAINS""{"Attributes "}" | empty

VALUE NOTATION      ::=   value(VALUE OBJECT IDENTIFIER)

Attributes          ::=   AttributeTerm | AttributeTerm "," Attributes

AttributeTerm       ::=   Attribute | AttributeSet
Attribute           ::=   value(ATTRIBUTE)
AttributeSet        ::=   value(ATTRIBUTE-SET)

END


ATTRIBUTE MACRO ::=
BEGIN

TYPENOTATION        ::= AttributeSyntax Multivalued | empty

VALUENOTATION       ::= value (VALUE OBJECT IDENTIFIER)

AttributeSyntax     ::=   "WITH ATTRIBUTE-SYNTAX"  SyntaxChoice

Multivalued         ::=   "SINGLE VALUE" | "MULTI VALUE" | empty

SyntaxChoice        ::=   value(ATTRIBUTE-SYNTAX) Constraint | type MatchTypes
```

```
Constraint              ::=   "(" ConstraintAlternative ")" | empty
ConstraintAlternative   ::=   StringConstraint | IntegerConstraint
StringConstraint        ::=   "SIZE" "(" SizeConstraint ")"
SizeConstraint          ::=   SingleValue | Range
SingleValue             ::=   value (INTEGER)
Range                   ::=   value (INTEGER) ".." value (INTEGER)
IntegerConstraint       ::=   Range

MatchTypes              ::=   "MATCHES FOR"  Matches | empty
Matches                 ::=   Match Matches | Match
Match                   ::=   "EQUALITY" | "SUBSTRINGS" | "ORDERING"

END


ATTRIBUTE-SYNTAX MACRO ::=
BEGIN

    TYPENOTATION        ::=   Syntax  MatchTypes | empty
    VALUENOTATION       ::=   value (VALUE OBJECT IDENTIFIER)

    Syntax              ::=   type

    MatchTypes          ::=   "MATCHES FOR" Matches | empty
    Matches             ::=   Match  Matches | Match
    Match               ::=   "EQUALITY" | "SUBSTRINGS" | "ORDERING"

END

-- object classes --
Top ::=             OBJECT-CLASS
                        MUST CONTAIN     {objectClass }

Alias ::=           OBJECT-CLASS
                        SUBCLASS OF top
                        MUST CONTAIN     {aliasedObjectName }


-- attribute types --
ObjectClass         ::=   ATTRIBUTE
                          WITH ATTRIBUTE-SYNTAX objectIdentifierSyntax

AliasedObjectName   ::=   ATTRIBUTE
                          WITH ATTRIBUTE-SYNTAX distinguishedNameSyntax
                          SINGLE VALUE

-- attribute syntaxes --
ObjectIdentifierSyntax  ::=
    ATTRIBUTE-SYNTAX OBJECT IDENTIFIER
        MATCHES FOR  EQUALITY

DistinguishedNameSyntax   ::=
    ATTRIBUTE-SYNTAX DistinguishedName
        MATCHES FOR EQUALITY

END
```

ANNEX G - AUTHENTICATION FRAMEWORK IN ASN.1

This Annex is part of the Recommendation the ASN.1 module, "**AuthenticationFramework**".

This Annex includes all of the ASN.1 type, macro and value definitions contained in this Recommendation in the form of

---

```
AuthenticationFramework {joint-iso-ccitt ds(5) modules(1) authenticationFramework(7)}
DEFINITIONS ::=
BEGIN

EXPORTS     AlgorithmIdentifier, AuthorityRevocationList, CACertificate, Certificate,
            Certificates, CertificationPath, CertificateRevocationList, UserCertificate,
            CrossCertificatePair, UserPassword,
            ALGORITHM, ENCRYPTED, PROTECTED, SIGNATURE, SIGNED;

IMPORTS
        informationFramework, selectedAttributeTypes, upperBounds
                FROM UsefulDefinitions {joint-iso-ccitt ds(5) modules(1) usefulDefinitions(0)}

        Name, ATTRIBUTE, ATTRIBUTE-SYNTAX
                FROM InformationFramework informationFramework

        ub-user-password    FROM UpperBounds upperBounds;

-- types

Certificate ::=     SIGNED SEQUENCE {
                        version    [0]          Version DEFAULT v1988,
                        serialNumber            CertificateSerialNumber,
                        signature           AlgorithmIdentifier,
                        issuer                  Name,
                        validity                Validity,
                        subject                 Name,
                        subjectPublicKeyInfo    SubjectPublicKeyInfo}

Version ::=         INTEGER {v1988(0)}

CertificateSerialNumber ::=     INTEGER

Validity ::=        SEQUENCE{
                        notBefore               UTCTime,
                        notAfter                UTCTime}

SubjectPublicKeyInfo ::=  SEQUENCE{
                        algorithm               AlgorithmIdentifier,
                        subjectPublicKey        BIT STRING}

AlgorithmIdentifier ::= SEQUENCE {
                        algorithm               OBJECT IDENTIFIER,
                        parameters              ANY DEFINED BY algorithm OPTIONAL}
```

```
Certificates      ::=  SEQUENCE {
                           certificate          Certificate,
                           certificationPath    ForwardCertificationPath  OPTIONAL}

ForwardCertificationPath ::= SEQUENCE OF CrossCertificates

CertificationPath      ::=  SEQUENCE {
                           userCertificate      Certificate,
                           theCACertificates    SEQUENCE OF CertificatePair OPTIONAL}

CrossCertificates      ::=  SET OF Certificate

CertificateList  ::=  SIGNED SEQUENCE {
                           signature            AlgorithmIdentifier,
                           issuer               Name,
                           lastUpdate                UTCTime,
                           revokedCertificates       SIGNED SEQUENCE OF SEQUENCE {
                                                     signature           AlgorithmIdentifier,
                                                     issuer              Name,
                                                     subject             CertificateSerialNumber,
                                                     revokationDate      UTCTime}
                                                          OPTIONAL}

CertificatePair  ::=  SEQUENCE {
                           forward    [0]   Certificate OPTIONAL,
                           reverse    [1]   Certificate OPTIONAL
                           -- at least one of the pair must be present -- }

-- attribute types

UserCertificate  ::=  ATTRIBUTE
                           WITH ATTRIBUTE-SYNTAX Certificate

CACertificate    ::=  ATTRIBUTE
                           WITH ATTRIBUTE-SYNTAX Certificate

CrossCertificatePair::=    ATTRIBUTE
                           WITH ATTRIBUTE-SYNTAX CertificatePair

CertificateRevocationList  ::= ATTRIBUTE
                           WITH ATTRIBUTE-SYNTAX CertificateList

AuthorityRevocationList  ::= ATTRIBUTE
                           WITH ATTRIBUTE-SYNTAX CertificateList

UserPassword  ::=  ATTRIBUTE
                           WITH ATTRIBUTE-SYNTAX
                                OCTET STRING (SIZE(0..ub-user-password))

-- macros

ALGORITHM MACRO ::=
BEGIN
TYPE NOTATION          ::=  "PARAMETER" type
VALUE NOTATION         ::=  value(VALUE OBJECT IDENTIFIER)
END -- of ALGORITHM
```

```
ENCRYPTED MACRO      ::=
BEGIN
TYPE NOTATION        ::=  type (ToBeEnciphered)
VALUE NOTATION       ::=  value (VALUE BIT STRING)
          -- the value of the bit string is generated by taking the octets which form the complete
          -- encoding (using the ASN.1 Basic Encoding Rules) of the value of the
          -- 'ToBeEnciphered' type and applying an encipherment procedure to those octets --}
                           )
END -- of ENCRYPTED

SIGNED MACRO     ::=
BEGIN
TYPE NOTATION        ::=  type (ToBeSigned)
VALUE NOTATION       ::=  value (VALUE
    SEQUENCE {
          ToBeSigned,
          AlgorithmIdentifier, -- of the algorithm used to generate the signature
          ENCRYPTED OCTET STRING
          -- where the octet string is the result of the hashing of the value of 'ToBeSigned'- -}
                           )
END- - of SIGNED

SIGNATURE MACRO ::=
BEGIN
TYPE NOTATION        ::=  type (OfSignature)
VALUE NOTATION       ::=  value (VALUE
    SEQUENCE {
          AlgorithmIdentifier,
          -- of the algorithm used to compute the signature
          ENCRYPTED OCTET STRING
          -- where the octet string is a function (e.g. a compressed or hashed version) of the
          --value 'OfSignature', which may include the identifier of the algorithm used to
          -- compute the signature - -}
                           )
END -- of SIGNATURE

PROTECTED MACRO ::= SIGNATURE

END -- of Authentication Framework Definitions
```

ANNEX A - ABSTRACT SERVICE IN ASN.1

This Annex is part of the standard.

This Annex includes all of the ASN.1 type, value and macro definitions contained in this Recommendation in the form of the ASN.1 module "**DirectoryAbstractService**".

```
DirectoryAbstractService {joint-ISO-CCITT ds(5) modules(1) directoryAbstractService(2)}
DEFINITIONS ::=
BEGIN

EXPORTS
    directory, readPort, searchPort, modifyPort,
    DirectoryBind, DirectoryBindArgument,
    DirectoryUnbind,
    Read, ReadArgument, ReadResult,
    Abandon, AbandonArgument, AbandonResult,
    Compare, CompareArgument, CompareResult,
    List, ListArgument, ListResult,
    Search, SearchArgument, SearchResult,
    AddEntry, AddEntryArgument, AddEntryResult,
    RemoveEntry, RemoveEntryArgument, RemoveEntryResult,
    ModifyEntry, ModifyEntryArgument, ModifyEntryResult,
    ModifyRDN, ModifyRDNArgument, ModifyRDNResult,
    Abandoned, AbandonFailed, AttributeError, NameError,
    Referral, SecurityError, ServiceError, UpdateError,
    SecurityParameters;

IMPORTS
    informationFramework, authenticationFramework, distributedOperations, directoryObjectIdentifiers
        FROM  UsefulDefinitions {joint-iso-ccitt ds(5) modules(1) usefulDefinitions(0)}

    OBJECT, PORT, ABSTRACT-BIND, ABSTRACT-UNBIND,
    ABSTRACT-OPERATION, ABSTRACT-ERROR
        FROM AbstractServiceNotation {joint-iso-ccitt mhs-motis(6) asdc(2) modules(0) notation(1) }

    Attribute, AttributeType, AttributeValue, AttributeValueAssertion,
    DistinguishedName, Name, RelativeDistinguishedName
        FROM   InformationFramework informationFramework

    id-ot-directory, id-ot-dua, id-pt-read, id-pt-search, id-pt-modify
        FROM DirectoryObjectIdentifiers directoryObjectIdentifiers

    ContinuationReference, OperationProgress
        FROM DistributedOperations distributedOperations

    Certificate, CertificationPath, SIGNED,
    PROTECTED, AlgorithmIdentifier
        FROM  AuthenticationFramework authenticationFramework
    InvokeID,
        FROM  Remote-Operations-Notation {joint-iso-ccitt remoteOperations(4) notation(0)};
```

```
-- macro for representing optional signing --
OPTIONALLY-SIGNED MACRO ::=
BEGIN
    TYPE NOTATION      ::= type (Type)
    VALUE NOTATION     ::= value (VALUE CHOICE { Type, SIGNED Type})
END

-- objects and ports --
directory OBJECT
    PORTS { readPort [S], searchPort [S], modifyPort [S] }
    ::=  id-ot-directory

dua OBJECT
    PORTS { readPort [C], searchPort [C], modifyPort [C]}
    ::=  id-ot-dua

readPort PORT
    CONSUMER INVOKES { Read, Compare, Abandon }
    ::=  id-pt-read

searchPort PORT
    CONSUMER INVOKES { List, Search }
    ::=  id-pt-search

modifyPort PORT
    CONSUMER INVOKES { AddEntry, RemoveEntry, ModifyEntry, ModifyRDN }
    ::=  id-pt-modify


-- bind and unbind --
DirectoryBind ::= ABSTRACT-BIND
    TO { readPort, searchPort, modifyPort }
    BIND
    ARGUMENT     DirectoryBindArgument
    RESULT       DirectoryBindResult
    BIND-ERROR   DirectoryBindError

DirectoryBindArgument   ::=  SET {
    credentials          [0]  Credentials OPTIONAL,
    versions             [1]  Versions DEFAULT {v1988}}

Credentials    ::=  CHOICE {
    simple               [0]  SimpleCredentials,
    strong               [1]  StrongCredentials,
    externalProcedure    [2]  EXTERNAL}

SimpleCredentials ::=     SEQUENCE{
    name      [0]  DistinguishedName,
    validity  [1]  SET{
                       time1     [0]  UTCTime OPTIONAL,
                       time2     [1]  UTCTime OPTIONAL,
                       random1   [2]  BIT STRING OPTIONAL,
                       random2   [3]  BIT STRING OPTIONAL} OPTIONAL,
    password  [2]  OCTET STRING OPTIONAL}

StrongCredentials ::=     SET {
    certification-path   [0]  CertificationPath OPTIONAL,
    bind-token           [1]  Token }
```

```
Token    ::=  SIGNED SEQUENCE {
    algorithm              [0]  AlgorithmIdentifier,
    name                   [1]  DistinguishedName,
    time                   [2]  UTCTime,
    random                 [3]  BIT STRING }

Versions ::=  BIT STRING {v1988(0)}

DirectoryBindResult ::=  DirectoryBindArgument

DirectoryBindError  ::=  SET {
    versions  [0] Versions DEFAULT {v1988},
    CHOICE {
        serviceError     [1]  ServiceProblem,
        securityError    [2]  SecurityProblem }}

DirectoryUnbind ::= ABSTRACT-UNBIND
    FROM {readPort, searchPort, modifyPort }

-- operations, arguments, and results --

Read ::= ABSTRACT-OPERATION
    ARGUMENT     ReadArgument
    RESULT       ReadResult
    ERRORS {
        AttributeError, NameError,
        ServiceError, Referral, Abandoned,
        SecurityError }

ReadArgument ::=  OPTIONALLY-SIGNED SET {
        object           [0]  Name,
        selection        [1]  EntryInformationSelection DEFAULT {},
        COMPONENTS OF CommonArguments }

ReadResult   ::=  OPTIONALLY-SIGNED SET {
        entry            [0]  EntryInformation,
        COMPONENTS OF CommonResults }

Compare ::= ABSTRACT-OPERATION
    ARGUMENT     CompareArgument
    RESULT       CompareResult
    ERRORS {
        AttributeError, NameError,
        ServiceError, Referral, Abandoned,
        SecurityError }

CompareArgument  ::=  OPTIONALLY-SIGNED SET {
    object               [0]  Name,
    purported            [1]  AttributeValueAssertion,
    COMPONENTS OF CommonArguments }

CompareResult    ::=  OPTIONALLY-SIGNED SET {
    DistinguishedName OPTIONAL,
    matched              [0]  BOOLEAN,
    fromEntry            [1]  BOOLEAN DEFAULT TRUE,
    COMPONENTS OF CommonResults }

Abandon ::= ABSTRACT-OPERATION
    ARGUMENT     AbandonArgument
    RESULT       AbandonResult
    ERRORS {AbandonFailed}

AbandonArgument ::=  SEQUENCE {
    invokeID  [0]  InvokeID}

AbandonResult ::= NULL
```

```
List ::= ABSTRACT-OPERATION
    ARGUMENT      ListArgument
    RESULT        ListResult
    ERRORS {
        NameError, ServiceError, Referral,
        Abandoned, SecurityError }

ListArgument  ::=  OPTIONALLY-SIGNED SET {
    object      [0]  Name,
    COMPONENTS OF CommonArguments }

ListResult      ::=  OPTIONALLY-SIGNED CHOICE{

    listInfo SET {
        DistinguishedName OPTIONAL,
        subordinates    [1]
                SET OF SEQUENCE {
                    RelativeDistinguishedName,
                    aliasEntry [0]  BOOLEAN DEFAULT FALSE,
                    fromEntry       [1]    BOOLEAN DEFAULT TRUE},
        partialOutcomeQualifier    [2]
                        PartialOutcomeQualifier OPTIONAL,
        COMPONENTS OF CommonResults},

    uncorrelatedListInfo [0] SET OF ListResult}

PartialOutcomeQualifier ::= SET {
    limitProblem                    [0]  LimitProblem OPTIONAL,
    unexplored                      [1]  SET OF ContinuationReference OPTIONAL,
    unavailableCriticalExtensions   [2]  BOOLEAN DEFAULT FALSE}

LimitProblem ::= INTEGER {
    timeLimitExceeded (0),
    sizeLimitExceeded (1),
    administrativeLimitExceeded (2) }

Search ::= ABSTRACT-OPERATION
    ARGUMENT      SearchArgument
    RESULT        SearchResult
    ERRORS {
        AttributeError, NameError,
        ServiceError, Referral, Abandoned,
        SecurityError }

SearchArgument      ::=  OPTIONALLY-SIGNED SET {
    baseObject      [0]  Name,
    subset          [1]  INTEGER {
                            baseObject      (0),
                            oneLevel        (1),
                            wholeSubtree    (2) }   DEFAULT baseObject,
    filter      [2]  Filter DEFAULT and{},
    searchAliases   [3]  BOOLEAN DEFAULT TRUE,
    selection       [4]  EntryInformationSelection DEFAULT {},
    COMPONENTS OF CommonArguments }

SearchResult  ::=  OPTIONALLY-SIGNED CHOICE{

    searchInfo SET {
        DistinguishedName OPTIONAL,
        entries                     [0]  SET OF EntryInformation,
        partialOutcomeQualifier     [2]  PartialOutcomeQualifier OPTIONAL,
        COMPONENTS OF CommonResults },

    uncorrelatedSearchInfo [0] SET OF SearchResult}
```

```
AddEntry ::=   ABSTRACT-OPERATION
    ARGUMENT     AddEntryArgument
    RESULT  AddEntryResult
    ERRORS{
         AttributeError, NameError,
         ServiceError, Referral, SecurityError,
         UpdateError}

AddEntryArgument  ::=  OPTIONALLY-SIGNED SET {
         object          [0]  DistinguishedName,
         entry           [1]  SET OF Attribute,
         COMPONENTS OF CommonArguments}

AddEntryResult          ::=  NULL

RemoveEntry ::=      ABSTRACT-OPERATION
    ARGUMENT     RemoveEntryArgument
    RESULT       RemoveEntryResult
    ERRORS {
         NameError, ServiceError, Referral,
         SecurityError, UpdateError}

RemoveEntryArgument    ::=  OPTIONALLY-SIGNED SET {
         object          [0]  DistinguishedName,
         COMPONENTS OF CommonArguments }

RemoveEntryResult ::= NULL

ModifyEntry ::=      ABSTRACT-OPERATION
    ARGUMENT     ModifyEntryArgument
    RESULT       ModifyEntryResult
    ERRORS {
         AttributeError, NameError,
         ServiceError, Referral, SecurityError,
         UpdateError}

ModifyEntryArgument     ::=  OPTIONALLY-SIGNED SET {
    object               [0]  DistinguishedName,
    changes              [1]  SEQUENCE OF EntryModification,
    COMPONENTS OF CommonArguments }

ModifyEntryResult ::= NULL

EntryModification ::= CHOICE {
    addAttribute         [0]  Attribute,
    removeAttribute      [1]  AttributeType,
    addValues            [2]  Attribute,
    removeValues         [3]  Attribute}

ModifyRDN ::= ABSTRACT-OPERATION
    ARGUMENT     ModifyRDNArgument
    RESULT       ModifyRDNResult
    ERRORS {
         NameError, ServiceError, Referral,
         SecurityError, UpdateError }

ModifyRDNArgument ::= OPTIONALLY-SIGNED SET {
    object               [0]  DistinguishedName,
    newRDN               [1]  RelativeDistinguishedName,
    deleteOldRDN         [2]  BOOLEAN DEFAULT FALSE,
    COMPONENTS OF CommonArguments }

ModifyRDNResult::= NULL
```

```
-- errors and parameters --
Abandoned ::= ABSTRACT-ERROR -- not literally an "error"

AbandonFailed ::= ABSTRACT-ERROR
    PARAMETER SET {
        problem   [0]  AbandonProblem,
        operation [1]  InvokeID}

AbandonProblem ::=      INTEGER {
                            noSuchOperation (1),
                            tooLate (2),
                            cannotAbandon (3) }

AttributeError ::= ABSTRACT-ERROR
    PARAMETER SET {
        object    [0]  Name,
        problems  [1]  SET OF SEQUENCE {
            problem   [0]  AttributeProblem,
            type      [1]  AttributeType,
            value     [2]  AttributeValue OPTIONAL }}

AttributeProblem    ::=  INTEGER {
                            noSuchAttributeOrValue (1),
                            invalidAttributeSyntax (2),
                            undefinedAttributeType (3),
                            inappropriateMatching (4),
                            constraintViolation (5),
                            attributeOrValueAlreadyExists(6)}

NameError ::= ABSTRACT-ERROR
    PARAMETER SET {
        problem   [0]  NameProblem,
        matched   [1]  Name}

NameProblem ::=         INTEGER {
                            noSuchObject (1),
                            aliasProblem (2),
                            invalidAttributeSyntax (3),
                            aliasDereferencingProblem (4) }

Referral ::= ABSTRACT-ERROR -- not.literally an "error"
    PARAMETER SET {
        candidate [0]  ContinuationReference}

SecurityError ::= ABSTRACT-ERROR
    PARAMETER SET {
        problem   [0]  SecurityProblem }

SecurityProblem     ::=  INTEGER {
                            inappropriateAuthentication (1),
                            invalidCredentials (2),
                            insufficientAccessRights (3),
                            invalidSignature (4),
                            protectionRequired (5),
                            noInformation (6) }

ServiceError ::= ABSTRACT-ERROR
    PARAMETER SET {
        problem   [0]  ServiceProblem}
```

```
ServiceProblem      ::=  INTEGER {
                             busy(1),
                             unavailable (2),
                             unwillingToPerform (3),
                             chainingRequired (4),
                             unableToProceed (5),
                             invalidReference (6),
                             timeLimitExceeded (7),
                             administrativeLimitExceeded (8),
                             loopDetected (9),
                             unavailableCriticalExtension (10),
                             outOfScope (11),
                             ditError (12) }

UpdateError ::= ABSTRACT-ERROR
    PARAMETER SET {
        problem   [0]  UpdateProblem }

UpdateProblem       ::=  INTEGER {
                             namingViolation (1),
                             objectClassViolation (2),
                             notAllowedOnNonLeaf (3),
                             notAllowedOnRDN (4),
                             entryAlreadyExists (5),
                             affectsMultipleDSAs (6),
                             objectClassModificationProhibited (7)}

-- common arguments/results --
CommonArguments   ::=  SET {
                             [30] ServiceControls DEFAULT {},
                             [29] SecurityParameters DEFAULT {},
    requestor                [28] DistinguishedName OPTIONAL,
                             [27] OperationProgress DEFAULT {notStarted},
    aliasedRDNs              [26] INTEGER OPTIONAL,
    extensions               [25] SET OF Extension OPTIONAL}

Extension      ::=  SET {
    identifier               [0]  INTEGER,
    critical                 [1]  BOOLEAN DEFAULT FALSE,
    item                     [2]  ANY DEFINED BY identifier }

CommonResults ::=  SET {
                             [30] SecurityParameters        OPTIONAL,
    performer                [29] DistinguishedName OPTIONAL,
    aliasDereferenced        [28] BOOLEAN DEFAULT FALSE}

-- common data types --
ServiceControls     ::=  SET {
    options          [0]  BIT STRING {
                             preferChaining(0),
                             chainingProhibited (1),
                             localScope (2),
                             dontUseCopy (3),
                             dontDereferenceAliases(4)}   DEFAULT {},
```

```
    priority          [1]  INTEGER {
                               low (0),
                               medium (1),
                               high (2) } DEFAULT medium,
    timeLimit         [2]  INTEGER OPTIONAL,
    sizeLimit         [3]  INTEGER OPTIONAL,

    scopeOfReferral   [4]  INTEGER{
                               dmd(0),
                               country(1)} OPTIONAL }

EntryInformationSelection::=   SET {
    attributeTypes    CHOICE {
                      allAttributes     [0]  NULL,
                      select            [1]  SET OF AttributeType - - empty set implies no attributes are requested - - }
                               DEFAULT allAttributes NULL,
    infoTypes                  [2]  INTEGER {
                                          attributeTypesOnly (0),
                                          attributeTypesAndValues (1) }
                               DEFAULT attributeTypesAndValues }

EntryInformation      ::=  SEQUENCE {
                      DistinguishedName,
    fromEntry         BOOLEAN DEFAULT TRUE,
                      SET OF CHOICE {AttributeType, Attribute} OPTIONAL }

Filter      ::=   CHOICE {
    item          [0]  FilterItem,
    and           [1]  SET OF Filter,
    or            [2]  SET OF Filter,
    not           [3]  Filter }
FilterItem        ::=  CHOICE {
    equality                [0]  AttributeValueAssertion,
    substrings              [1]  SEQUENCE {
                                     type        AttributeType,
                                     strings SEQUENCE OF CHOICE {
                                                 initial  [0]    AttributeValue,
                                                 any      [1]    AttributeValue,
                                                 final    [2]    AttributeValue}},
    greaterOrEqual          [2]  AttributeValueAssertion,
    lessOrEqual             [3]  AttributeValueAssertion,
    present                 [4]  AttributeType,
    approximateMatch        [5]  AttributeValueAssertion }

SecurityParameters  ::=  SET {
    certification-path      [0]  CertificationPath OPTIONAL,
    name                    [1]  DistinguishedName OPTIONAL,
    time                    [2]  UTCTime OPTIONAL,
    random                  [3]  BIT STRING OPTIONAL,
    target                  [4]  ProtectionRequest OPTIONAL}

ProtectionRequest   ::=  INTEGER {
                               none (0),
                               signed (1)}

CertificationPath     ::=  SEQUENCE OF Certificate

END
```

ANNEX A - ASN.1 FOR DISTRIBUTED OPERATIONS

This Annex is part of the standard.

This Annex includes all of the ASN.1 type, value and macro definitions contained in this Recommendation in the form of the ASN.1 module "**DistributedOperations**".

```
DistributedOperations    {joint-iso-ccitt ds(5) modules(1) distributedOperations(3)}
DEFINITIONS ::=
BEGIN

EXPORTS
    DirectoryRefinement, chainedReadPort, chainedSearchPort, chainedModifyPort,
    DSABind, DSABindArgument,
    DSAUnbind,
    ChainedRead, ChainedCompare, ChainedAbandon,
    ChainedList, ChainedSearch,
    ChainedAddEntry, ChainedRemoveEntry,
    ChainedModifyEntry, ChainedModifyRDN,
    DsaReferral, ContinuationReference;

IMPORTS
    informationFramework, abstractService, distributedOperations,
    directoryObjectIdentifiers, selectedAttributeTypes
        FROM  UsefulDefinitions {joint-iso-ccitt ds(5) modules(1) usefulDefinitions(0)}

    DistinguishedName, Name, RelativeDistinguishedName
        FROM  InformationFramework informationFramework

    id-ot-dsa, id-pt-chained-read, id-pt-chained-search, id-pt-chained-modify
        FROM  DistributedDirectoryObjectIdentifiers distributedDirectoryObjectIdentifiers

    PresentationAddress
        FROM SelectedAttributeTypes selectedAttributeTypes

    directory, readPort, searchPort, modifyPort
    DirectoryBind,
    ReadArgument, ReadResult,
    CompareArgument, CompareResult,
    Abandon
    ListArgument, ListResult,
    SearchArgument, SearchResult,
    AddEntryArgument, AddEntryResult,
    RemoveEntryArgument, RemoveEntryResult,
    ModifyEntryArgument, ModifyEntryResult,
    ModifyRDNArgument, ModifyRDNResult,
    Abandoned, AttributeError, NameError, ServiceError, SecurityError, UpdateError
    OPTIONALLY-SIGNED,
        FROM DirectoryAbstractService directoryAbstractService;
```

```
-- objects and ports --
DirectoryRefinement ::=  REFINE directory AS
    dsa       RECURRING
          readPort              [S]  VISIBLE
          searchPort            [S]  VISIBLE
          modifyPort            [S]  VISIBLE
          chainedReadPort       PAIRED WITH dsa
          chainedSearchPort     PAIRED WITH dsa
          chainedModifyPort     PAIRED WITH dsa

dsa OBJECT
          PORTS {  readPort        [S],
                   searchPort      [S],
                   modifyPort      [S],
                   chainedReadPort,
                   chainedSearchPort,
                   chainedModifyPort  }
    ::=  id-ot-dsa

chainedReadPort  PORT
    ABSTRACT OPERATIONS {
          ChainedRead, ChainedCompare, ChainedAbandon}
    ::=  id-pt-chained-read

chainedSearchPort  PORT
    ABSTRACT OPERATIONS{
          ChainedList, ChainedSearch}
    ::=  id-pt-chained-search

chainedModifyPort  PORT
    ABSTRACT OPERATIONS {
          ChainedAddEntry, ChainedRemoveEntry,
          ChainedModifyEntry, ChainedModifyRDN}
    ::=  id-pt-chained-modify

DSABind  ::=  ABSTRACT-BIND
    TO        {chainedRead,
              chainedSearch,
              chainedModify}
    DirectoryBind

DSAUnbind  ::= ABSTRACT-UNBIND
    FROM          {chainedRead,
                  chainedSearch,
                  chainedModify}

-- operations, arguments, and results --
ChainedRead  ::=
    ABSTRACT-OPERATION
          ARGUMENT      OPTIONALLY-SIGNED
               SET{
                    ChainingArguments,
                    [0]  ReadArgument}
          RESULT        OPTIONALLY-SIGNED
               SET{
                    ChainingResults,
                    [0]  ReadResult}
          ERRORS {
               DsaReferral, Abandoned, AttributeError, NameError,
               ServiceError, SecurityError}
```

```
ChainedCompare    ::=
    ABSTRACT-OPERATION
        ARGUMENT    OPTIONALLY-SIGNED
            SET{
                ChainingArguments,
                [0]  CompareArgument}
        RESULT      OPTIONALLY-SIGNED
            SET{
                ChainingResults,
                [0]  CompareResult}
        ERRORS {
            DsaReferral, Abandoned, AttributeError, NameError,
            ServiceError, SecurityError}

ChainedAbandon  ::= Abandon

ChainedList    ::=
    ABSTRACT-OPERATION
        ARGUMENT    OPTIONALLY-SIGNED
            SET{
                ChainingArguments,
                [0]  ListArgument}
        RESULT      OPTIONALLY-SIGNED
            SET{
                ChainingResults,
                [0]  ListResult}
        ERRORS {
            DsaReferral, Abandoned, AttributeError, NameError,
            ServiceError, SecurityError}

ChainedSearch      ::=
    ABSTRACT-OPERATION
        ARGUMENT    OPTIONALLY-SIGNED
            SET{
                ChainingArguments,
                [0]  SearchArgument}
        RESULT      OPTIONALLY-SIGNED
            SET{
                ChainingResults,
                [0]  SearchResult}
        ERRORS {
            DsaReferral, Abandoned, AttributeError, NameError,
            ServiceError, SecurityError}

ChainedAddEntry    ::=
    ABSTRACT-OPERATION
        ARGUMENT    OPTIONALLY-SIGNED
            SET{
                ChainingArguments,
                [0]  AddEntryArgument}
        RESULT      OPTIONALLY-SIGNED
            SET{
                ChainingResults,
                [0]  AddEntryResult}
        ERRORS {
            DsaReferral, Abandoned, AttributeError, NameError,
            ServiceError, SecurityError, UpdateError}
```

```
ChainedRemoveEntry      ::=
    ABSTRACT-OPERATION
        ARGUMENT      OPTIONALLY-SIGNED
            SET{
                ChainingArguments,
                [0]   RemoveEntryArgument}
        RESULT        OPTIONALLY-SIGNED
            SET{
                ChainingResults,
                [0]   RemoveEntryResult}
        ERRORS {
            DsaReferral, Abandoned, NameError,
            ServiceError, SecurityError, UpdateError}

ChainedModifyEntry      ::=
    ABSTRACT-OPERATION
        ARGUMENT      OPTIONALLY-SIGNED
            SET{
                ChainingArguments,
                [0]   ModifyEntryArgument}
        RESULT        OPTIONALLY-SIGNED
            SET{
                ChainingResults,
                [0]   ModifyEntryResult}
        ERRORS {
            DsaReferral, Abandoned, AttributeError, NameError,
            ServiceError, SecurityError, UpdateError}

ChainedModifyRDN  ::=
    ABSTRACT-OPERATION
        ARGUMENT      OPTIONALLY-SIGNED
            SET{
                ChainingArguments,
                [0]   ModifyRDNArgument}
        RESULT        OPTIONALLY-SIGNED
            SET{
                ChainingResults,
                [0]   ModifyRDNResult}
        ERRORS {
            DsaReferral, Abandoned, NameError,
            ServiceError, SecurityError, UpdateError}

-- errors and parameters --

DSAReferral   ::=
    ABSTRACT-ERROR
        PARAMETER SET{
                              [0]   ContinuationReference,
            contextPrefix     [1]   DistinguishedName OPTIONAL }
```

```
-- common arguments/results --

ChainingArguments ::=   SET {
    originator              [0]  DistinguishedName  OPTIONAL,
    targetObject            [1]  DistinguishedName  OPTIONAL,
    operationProgress       [2]  OperationProgress  DEFAULT {notStarted},
    traceInformation        [3]  TraceInformation,
    aliasDereferenced       [4]  BOOLEAN DEFAULT FALSE,
    aliasedRDNs             [5]  INTEGER  OPTIONAL,
        -- absent unless aliasDereferenced is TRUE
    returnCrossRefs         [6]  BOOLEAN DEFAULT FALSE,
    referenceType           [7]  ReferenceType DEFAULT superior,
    info                    [8]  DomainInfo  OPTIONAL,
    timeLimit               [9]  UTCTime  OPTIONAL,
                            [10] SecurityParameters DEFAULT {}}

ChainingResults     ::= SET {
    info                    [0]  DomainInfo  OPTIONAL,
    crossReferences         [1]  SEQUENCE OF CrossReference  OPTIONAL,
                            [2]  SecurityParameters DEFAULT {}}

CrossReference      ::=  SET{
    contextPrefix           [0]  DistinguishedName,
    accessPoint             [1]  AccessPoint}

ReferenceType       ::=  ENUMERATED {
    superior                     (1),
    subordinate                  (2),
    cross                        (3),
    nonSpecificSubordinate       (4)}

TraceInformation    ::=  SEQUENCE OF TraceItem

TraceItem           ::=  SET {
    dsa                     [0]  Name,
    targetObject            [1]  Name  OPTIONAL,
    operationProgress       [2]  OperationProgress}

OperationProgress   ::=  SET {
    nameResolutionPhase     [0]  ENUMERATED {notStarted (1), proceeding (2), completed (3)},
    nextRDNToBeResolved     [1]  INTEGER  OPTIONAL}

DomainInfo    ::=  ANY

ContinuationReference   ::=  SET {
    targetObject            [0]  Name,
    aliasedRDNs             [1]  INTEGER  OPTIONAL,
    operationProgress       [2]  OperationProgress,
    rdnsResolved            [3]  INTEGER  OPTIONAL,
    referenceType           [4]  ReferenceType  OPTIONAL,
                            -- only present in the DSP
    accessPoints            [5]  SET OF AccessPoint}

AccessPoint   ::=   SET {
    ae-title          [0]  Name,
    address           [1]  PresentationAddress }

END
```

ANNEX A - DAP IN ASN.1

This Annex is part of the standard.

This Annex includes all of the ASN.1 type and value definitions contained in this Recommendation in the form of the ASN.1 module, **"DirectoryAccessProtocol"**.

```
DirectoryAccessProtocol {joint-iso-ccitt ds(5) modules(1) dap(11)}
DEFINITIONS ::=
BEGIN

EXPORTS
        directoryAccessAC, readASE, searchASE, modifyASE;

IMPORTS
    abstractService
    FROM UsefulDefinitions
        {joint-iso-ccitt ds(5) modules(1) usefulDefinitions(0)}

    APPLICATION-SERVICE-ELEMENT, APPLICATION-CONTEXT, aCSE
        FROM Remote-Operations-Notation-extension
            {joint-iso-ccitt remoteOperations(4) notation-extension(2)}

    id-ac-directoryAccessAC, id-ase-readASE, id-ase-searchASE,
    id-ase-modifyASE, id-as-directoryAccessAS, id-as-acse
        FROM ProtocolObjectIdentifiers
            {joint-iso-ccitt ds(5) modules(1) protocolObjectIdentifiers(4)}

    DirectoryBind, DirectoryUnbind, Read, Compare, Abandon, List, Search,
    AddEntry, RemoveEntry, ModifyEntry, ModifyRDN, Abandoned, AbandonFailed
    AttributeError, NameError, Referral, SecurityError, ServiceError, UpdateError
        FROM DirectoryAbstractService directoryabstractService;


-- Application Contexts --

directoryAccessAC
    APPLICATION-CONTEXT
        APPLICATION SERVICE ELEMENTS {aCSE}
        BIND DirectoryBind
        UNBIND DirectoryUnbind
        REMOTE OPERATIONS {rOSE}
            INITIATOR CONSUMER OF {readASE, searchASE, modifyASE}
        ABSTRACT SYNTAXES {
            id-as-acse, id-as-directoryAccessAS}
        ::= id-ac-directoryAccessAC


-- Read ASE --

readASE
        APPLICATION-SERVICE-ELEMENT
            CONSUMER INVOKES { read, compare, abandon}
        ::= id-ase-readASE
```

```
-- Search ASE --
searchASE
        APPLICATION-SERVICE-ELEMENT
            CONSUMER INVOKES { list, search}
        ::= id-ase-searchASE
-- Modify ASE --
modifyASE
        APPLICATION-SERVICE-ELEMENT
            CONSUMER INVOKES
                {addEntry, removeEntry,
                modifyEntry, modifyRDN}
        ::= id-ase-modifyASE
-- Remote Operations --
read            Read            ::=  1
compare         Compare         ::=  2
abandon         Abandon         ::=  3
list            List            ::=  4
search          Search          ::=  5
addEntry        AddEntry        ::=  6
removeEntry     RemoveEntry     ::=  7
modifyEntry     ModifyEntry     ::=  8
modifyRDN       ModifyRDN       ::=  9
-- Remote Errors --
attributeError  AttributeError  ::=  1
nameError       NameError       ::=  2
serviceError    ServiceError    ::=  3
referral        Referral        ::=  4
abandoned       Abandoned       ::=  5
securityError   SecurityError   ::=  6
abandonFailed   AbandonFailed   ::=  7
updateError     UpdateError     ::=  8
END
```

ANNEX B - DSP IN ASN.1

This Annex is part of the standard.

This Annex includes all of the ASN.1 type and value definitions contained in this Recommendation in the form of the ASN.1 module, **"DirectorySystemProtocol"**.

```
DirectorySystemProtocol {joint-iso-ccitt ds(5) modules(1) dsp(12)}
DEFINITIONS ::=
BEGIN

EXPORTS
        directorySystemAC, chainedReadASE, chainedSearchASE, chainedModifyASE;

IMPORTS
    distributedOperations, directoryAbstractService
        FROM UsefulDefinitions {joint-iso-ccitt ds(5) modules(1) usefulDefinitions(0)}

    APPLICATION-SERVICE-ELEMENT, APPLICATION-CONTEXT, aCSE
        FROM Remote-Operations-Notation-extension
            {joint-iso-ccitt remoteOperations(4) notation-extension(2)}

    id-ac-directorySystemAC, id-ase-chainedReadASE, id-ase-chainedSearchASE,
    id-ase-chainedModifyASE, id-as-directorySystemAS, id-as-acse;
        FROM ProtocolObjectIdentifiers
            {joint-iso-ccitt ds(5) modules(1) protocolObjectIdentifiers(4)}

    Abandoned, AbandonFailed, AttributeError,
    NameError, SecurityError, ServiceError, UpdateError
        FROM DirectoryAbstractService directoryAbstractService

    DSABind, DSAUnbind,
    ChainedRead, ChainedCompare, ChainedAbandon,
    ChainedList, ChainedSearch,
    ChainedAddEntry, ChainedRemoveEntry, ChainedModifyEntry, ChainedModifyRDN,
    DSAReferral
        FROM DistributedOperations distributedOperations;

- - Application Contexts - -

directorySystemAC
    APPLICATION-CONTEXT
        APPLICATION SERVICE ELEMENTS {aCSE}
        BIND DSABind
        UNBIND DSAUnbind
        REMOTE OPERATIONS {rOSE}
            OPERATIONS OF {
                chainedReadASE, chainedSearchASE, chainedModifyASE}
        ABSTRACT SYNTAXES {
            id-as-acse, id-as-directorySystemAS}
        ::= {id-ac-directorySystemAC}
```

```
-- Chained Read ASE --
chainedReadASE
        APPLICATION-SERVICE-ELEMENT
             OPERATIONS { chainedRead, chainedCompare, chainedAbandon}
        ::=  id-ase-chainedReadASE

-- Chained Search ASE --
chainedSearchASE
        APPLICATION-SERVICE-ELEMENT
             OPERATIONS { chainedList, chainedSearch}
        ::=  id-ase-chainedSearchASE

-- Chained Modify ASE --
chainedModifyASE
        APPLICATION-SERVICE-ELEMENT
             OPERATIONS
                  {chainedAddEntry, chainedRemoveEntry,
                  chainedModifyEntry, chainedModifyRDN}
        ::=  id-ase-chainedModifyASE

-- Remote Operations --
chainedRead             ChainedRead             ::=  1
chainedCompare          ChainedCompare          ::=  2
chainedAbandon          ChainedAbandon          ::=  3
chainedList             ChainedList             ::=  4
chainedSearch           ChainedSearch           ::=  5
chainedAddEntry         ChainedAddEntry         ::=  6
chainedRemoveEntry      ChainedRemoveEntry      ::=  7
chainedModifyEntry      ChainedModifyEntry      ::=  8
chainedModifyRDN        ChainedModifyRDN        ::=  9

-- Remote Errors --
attributeError      AttributeError      ::=  1
nameError           NameError           ::=  2
serviceError        ServiceError        ::=  3
abandoned           Abandoned           ::=  5
securityError       SecurityError       ::=  6
abandonFailed       AbandonFailed       ::=  7
updateError         UpdateError         ::=  8
dsaReferral         DSAReferral         ::=  9
END
```

ANNEX A - SELECTED ATTRIBUTE TYPES IN ASN.1

This Annex is part of the standard.

This Annex includes all of the ASN.1 type and value definitions contained in this Recommendation in the form of the ASN.1 module, **"SelectedAttributeTypes"**.

```
SelectedAttributeTypes {joint-iso-ccitt ds(5) modules(1) selectedAttributeTypes(5)}
DEFINITIONS ::=
BEGIN

-- EXPORTS EVERYTHING

IMPORTS
    informationFramework, authenticationFramework, attributeType, upperBounds
        FROM UsefulDefinitions {joint-iso-ccitt ds(5) modules(1) usefulDefinitions(0) },
    ATTRIBUTE, ATTRIBUTE-SYNTAX, AttributeType, OBJECT-CLASS,
    ObjectClass, AliasedObjectName,
    DistinguishedNameSyntax, ObjectIdentifierSyntax
        FROM InformationFramework informationFramework
    G3FacsimileNonBasicParameters, TeletexNonBasicParameters
        FROM MTSAbstractService
                    {joint-iso-ccitt mhs-motis(6) mts(3) modules(0) mts-abstract-service(1)}
    UserCertificate, CACertificate, CrossCertificatePair, CertificateRevocationList,
    AuthorityRevocationList, UserPassword
        FROM AuthenticationFramework authenticationFramework
    ub-answerback, ub-common-name, ub-surname, ub-serial-number, ub-locality-name, ub-state-name,
    ub-street-address, ub-organization-name, ub-organizational-unit-name, ub-title, ub-description,
    ub-business-category, ub-postal-line, ub-postal-string, ub-postal-code, ub-post-office-box,
    ub-physical-office-name, ub-telex-number, ub-country-code, ub-teletex-terminal-id,
    ub-telephone-number, ub-x121-address, ub-international-isdn-number, ub-destination-indicator,
    ub-user-password
        FROM UpperBounds upperBounds;

-- attribute types --

    objectClass ObjectClass ::= {attributeType 0}

    aliasedObjectName AliasedObjectName ::= {attributeType 1}

    knowledgeInformation ATTRIBUTE
        WITH ATTRIBUTE-SYNTAX caseIgnoreStringSyntax
        ::= {attributeType 2}

    commonName ATTRIBUTE
        WITH ATTRIBUTE-SYNTAX
            caseIgnoreStringSyntax (SIZE(1..ub-common-name))
        ::= {attributeType 3}

    surname ATTRIBUTE
        WITH ATTRIBUTE-SYNTAX
            caseIgnoreStringSyntax (SIZE(1..ub-surname))
        ::= {attributeType 4}
```

```
serialNumber ATTRIBUTE
     WITH ATTRIBUTE-SYNTAX
          printableStringSyntax  (SIZE(1..ub-serial-number))
     ::= {attributeType 5}

countryName ATTRIBUTE
     WITH ATTRIBUTE-SYNTAX
          PrintableString (SIZE (2)) -- ISO 3166 codes only
     MATCHES FOR EQUALITY
     SINGLE VALUE
     ::= {attributeType 6}

localityName ATTRIBUTE
     WITH ATTRIBUTE-SYNTAX
          caseIgnoreStringSyntax  (SIZE(1..ub-locality-name))
     ::= {attributeType 7}

stateOrProvinceName ATTRIBUTE
     WITH ATTRIBUTE-SYNTAX
          caseIgnoreStringSyntax  (SIZE(1..ub-state-name))
     ::= {attributeType  8}

streetAddress ATTRIBUTE
     WITH ATTRIBUTE-SYNTAX
          caseIgnoreStringSyntax  (SIZE(1..ub-street-address))
     ::= {attributeType  9}

organizationName ATTRIBUTE
     WITH ATTRIBUTE-SYNTAX
          caseIgnoreStringSyntax  (SIZE(1..ub-organization-name))
     ::= {attributeType 10}

organizationalUnitName ATTRIBUTE
     WITH ATTRIBUTE-SYNTAX
          caseIgnoreStringSyntax  (SIZE(1..ub-organizational-unit-name))
     ::= {attributeType 11}

title ATTRIBUTE
     WITH ATTRIBUTE-SYNTAX
          caseIgnoreStringSyntax  (SIZE(1..ub-title))
     ::= {attributeType 12}

description ATTRIBUTE
     WITH ATTRIBUTE-SYNTAX
          caseIgnoreStringSyntax  (SIZE(1..ub-description))
     ::= {attributeType 13}

searchGuide ATTRIBUTE
     WITH ATTRIBUTE-SYNTAX
          Guide
     ::= {attributeType 14}

Guide     ::=  SET {
     objectClass     [0]  OBJECT-CLASS  OPTIONAL,
     criteria        [1]  Criteria}

Criteria  ::=
     CHOICE {
          type [0]  CriteriaItem,
          and  [1]  SET OF Criteria,
          or   [2]  SET OF Criteria,
          not  [3]  Criteria}
```

```
CriteriaItem    ::=   CHOICE {
     equality              [0]  AttributeType,
     substrings            [1]  AttributeType,
     greaterOrEqual        [2]  AttributeType,
     lessOrEqual           [3]  AttributeType,
     approximateMatch      [4]  AttributeType}

businessCategory ATTRIBUTE
     WITH ATTRIBUTE-SYNTAX
          caseIgnoreStringSyntax  (SIZE(1..ub-business-category))
     ::= {attributeType 15}

postalAddress ATTRIBUTE
     WITH ATTRIBUTE-SYNTAX PostalAddress
     MATCHES FOR EQUALITY
     ::= {attributeType 16}

PostalAddress ::= SEQUENCE SIZE(1..ub-postal-line) OF
     CHOICE {
          T61String  (SIZE(1..ub-postal-string)),
          PrintableString  (SIZE(1..ub-postal-string))}

postalCode ATTRIBUTE
     WITH ATTRIBUTE-SYNTAX
          caseIgnoreStringSyntax  (SIZE(1..ub-postal-code))
     ::= {attributeType 17}

postOfficeBox ATTRIBUTE
     WITH ATTRIBUTE-SYNTAX
          caseIgnoreStringSyntax  (SIZE(1..ub-post-office-box))
     ::= {attributeType  18}

physicalDeliveryOfficeName ATTRIBUTE
     WITH ATTRIBUTE-SYNTAX
          caseIgnoreStringSyntax  (SIZE(1..ub-physical-office-name))
     ::= {attributeType  19}

telephoneNumber ATTRIBUTE
     WITH ATTRIBUTE-SYNTAX
          telephoneNumberSyntax
     ::= {attributeType 20}

telexNumber ATTRIBUTE
     WITH ATTRIBUTE-SYNTAX TelexNumber
     ::= {attributeType 21}

TelexNumber ::= SEQUENCE {
     telexNumber     PrintableString
          (SIZE(1..ub-telex-number)),
     countryCode     PrintableString,
          (SIZE(1..ub-country-code)),
     answerback      PrintableString
          (SIZE(1..ub-answerback))}

teletexTerminalIdentifier ATTRIBUTE
     WITH ATTRIBUTE-SYNTAX
          TeletexTerminalIdentifier
     ::= {attributeType 22}

TeletexTerminalIdentifier ::= SEQUENCE {
     teletexTerminal  PrintableString  (SIZE(1..ub-teletex-terminal-id)),
     parameters  TeletexNonBasicParameters  OPTIONAL}
```

```
facsimileTelephoneNumber ATTRIBUTE
     WITH ATTRIBUTE-SYNTAX
          FacsimileTelephoneNumber
     ::= {attributeType 23}

FacsimileTelephoneNumber:= SEQUENCE {
     telephoneNumber PrintableString (SIZE(1.. ub-telephone-number)),
     parameters G3FacsimileNonBasicParameters   OPTIONAL}

x121Address ATTRIBUTE
     WITH ATTRIBUTE-SYNTAX
          NumericString (SIZE(1 .. ub-x121-address))
     MATCHES FOR EQUALITY SUBSTRINGS
     ::= {attributeType 24}

internationalISDNNumber ATTRIBUTE
     WITH ATTRIBUTE-SYNTAX
          NumericString (SIZE(1 .. ub-international-isdn-number))
     ::= {attributeType 25}

registeredAddress ATTRIBUTE
     WITH ATTRIBUTE-SYNTAX PostalAddress
     ::= {attributeType  26}

destinationIndicator ATTRIBUTE
     WITH ATTRIBUTE-SYNTAX
          PrintableString (SIZE(1.. ub-destination-indicator))
     -- alphabetical characters only
     MATCHES FOR EQUALITY SUBSTRINGS
     ::= {attributeType 27}

preferredDeliveryMethod ATTRIBUTE
     WITH ATTRIBUTE-SYNTAX
          SEQUENCE OF INTEGER {
               any-delivery-method        (0),
               mhs-delivery               (1),
               physical-delivery          (2),
               telex-delivery             (3),
               teletex-delivery           (4),
               g3-facsimile-delivery      (5),
               g4-facsimile-delivery      (6),
               ia5-terminal-delivery      (7),
               videotex-delivery          (8),
               telephone-delivery         (9)}
     SINGLE VALUE
     ::= {attributeType 28}

presentationAddress ATTRIBUTE
     WITH ATTRIBUTE-SYNTAX
          PresentationAddress
     MATCHES FOR EQUALITY
     SINGLE VALUE
     ::= {attributeType 29}

PresentationAddress ::= SEQUENCE {
     pSelector      [0]  OCTET STRING OPTIONAL,
     sSelector      [1]  OCTET STRING OPTIONAL,
     tSelector      [2]  OCTET STRING OPTIONAL,
     nAddresses     [3]  SET SIZE(1..MAX) OF OCTET STRING}
```

```
supportedApplicationContext  ATTRIBUTE
     WITH  ATTRIBUTE-SYNTAX
          objectIdentifierSyntax
     ::= {attributeType   30}

member ATTRIBUTE
     WITH  ATTRIBUTE-SYNTAX
          distinguishedNameSyntax
     ::= {attributeType 31}

owner ATTRIBUTE
     WITH  ATTRIBUTE-SYNTAX
          distinguishedNameSyntax
     ::= {attributeType 32}

roleOccupant  ATTRIBUTE
     WITH  ATTRIBUTE-SYNTAX
          distinguishedNameSyntax
     ::= {attributeType 33}

seeAlso  ATTRIBUTE
     WITH  ATTRIBUTE-SYNTAX
          distinguishedNameSyntax
     ::= {attributeType 34}

userPassword  UserPassword
     ::= {attributeType 35}

userCertificate  UserCertificate
     ::= {attributeType   36}

cACertificate  CACertificate
     ::= {attributeType   37}

authorityRevocationList  AuthorityRevocationList
     ::= {attributeType   38}

certificateRevocationList  CertificateRevocationList
     ::= {attributeType   39}

crossCertificatePair  CrossCertificatePair
     ::= {attributeType 40}


-- attribute syntaxes --

undefined  ATTRIBUTE-SYNTAX
     ANY
     ::= {attributeSyntax  0}

distinguishedNameSyntax    DistinguishedNameSyntax
     ::= {attributeSyntax  1}

objectIdentifierSyntax  ObjectIdentifierSyntax
     ::= {attributeSyntax  2}

caseExactStringSyntax  ATTRIBUTE-SYNTAX
     CHOICE  {T61String,  PrintableString}
     MATCHES  FOR  EQUALITY  SUBSTRINGS
     ::= {attributeSyntax  3}
```

```
caseIgnoreStringSyntax  ATTRIBUTE-SYNTAX
        CHOICE {T61String, PrintableString}
        MATCHES FOR EQUALITY SUBSTRINGS
        ::= {attributeSyntax 4}

printableStringSyntax  ATTRIBUTE-SYNTAX
        PrintableString
        MATCHES FOR EQUALITY SUBSTRINGS
        ::= {attributeSyntax 5}

numericStringSyntax  ATTRIBUTE-SYNTAX
        NumericString
        MATCHES FOR EQUALITY SUBSTRINGS
        ::= {attributeSyntax 6}

caseIgnoreListSyntax  ATTRIBUTE-SYNTAX
        SEQUENCE OF
            CHOICE {T61String, PrintableString}
        MATCHES FOR EQUALITY SUBSTRINGS
        ::= {attributeSyntax 7}

booleanSyntax  ATTRIBUTE-SYNTAX
        BOOLEAN
        MATCHES FOR EQUALITY
        ::= {attributeSyntax  8}

integerSyntax  ATTRIBUTE-SYNTAX
        INTEGER
        MATCHES FOR EQUALITY ORDERING
        ::= {attributeSyntax 9}

octetStringSyntax  ATTRIBUTE-SYNTAX
        OCTET STRING
        MATCHES FOR EQUALITY SUBSTRINGS ORDERING
        ::= {attributeSyntax 10}

uTCTimeSyntax  ATTRIBUTE-SYNTAX
        UTCTime
        MATCHES FOR EQUALITY ORDERING
        ::= {attributeSyntax 11}

telephoneNumberSyntax  ATTRIBUTE-SYNTAX
        PrintableString (SIZE(1 .. ub-telephone-number))
        MATCHES FOR EQUALITY SUBSTRINGS
        ::= {attributeSyntax 12}

END
```

ANNEX A - SELECTED OBJECT CLASSES IN ASN.1

This Annex is part of the standard.

This Annex includes all of the ASN.1 type and value definitions contained in this Recommendation in the form of the ASN.1 module, **"SelectedObjectClasses"**.

```
SelectedObjectClasses {joint-iso-ccitt ds(5) modules(1) selectedObjectClasses(6)}
DEFINITIONS ::=
BEGIN

- - EXPORTS EVERYTHING;

IMPORTS
    objectClass, attributeSet, informationFramework, selectedAttributeTypes
        FROM UsefulDefinitions {joint-iso-ccitt ds(5) modules(1) usefulDefinitions(0) }
    OBJECT-CLASS,ATTRIBUTE-SET, Top, Alias
        FROM InformationFramework informationFramework
    authorityRevocationList, businessCategory, cACertificate, certificateRevocationList, commonName,
    countryName, description, destinationIndicator, facsimileTelephoneNumber, internationalISDNNumber,
    knowledgeInformation, localityName, member, organizationName, organizationalUnitName,
    owner, physicalDeliveryOfficeName, postOfficeBox, postalAddress, postalCode,
    preferredDeliveryMethod, presentationAddress, registeredAddress, roleOccupant, searchGuide,
    seeAlso, serialNumber, stateOrProvinceName, streetAddress, supportedApplicationContext,
    surname, telephoneNumber, teletexTerminalIdentifier, telexNumber, title, userPassword, x121Address
        FROM SelectedAttributeTypes selectedAttributeTypes;


    telecommunicationAttributeSet ATTRIBUTE-SET
        CONTAINS {
            facsimileTelephoneNumber,
            internationalISDNNumber,
            telephoneNumber,
            teletexTerminalIdentifier,
            telexNumber,
            preferredDeliveryMethod,
            destinationIndicator,
            registeredAddress,
            x121Address}
            ::= {attributeSet 0}

    postalAttributeSet ATTRIBUTE-SET
        CONTAINS {
            physicalDeliveryOfficeName,
            postalAddress,
            postalCode,
            postOfficeBox,
            streetAddress}
            ::= {attributeSet 1}
```

```
localeAttributeSet  ATTRIBUTE-SET
     CONTAINS {
          localityName,
          stateOrProvinceName,
          streetAddress}
          ::= {attributeSet 2}

organizationalAttributeSet  ATTRIBUTE-SET
     CONTAINS {
          description,
          localeAttributeSet,
          postalAttributeSet,
          telecommunicationAttributeSet,
          businessCategory,
          seeAlso,
          searchGuide,
          userPassword}
          ::= {attributeSet 3}

top       Top       ::=  {objectClass 0}

alias     Alias     ::=  {objectClass 1}

country  OBJECT-CLASS
     SUBCLASS OF top
     MUST CONTAIN {
          countryName}
     MAY CONTAIN {
          description,
          searchGuide}
     ::= {objectClass 2}

locality  OBJECT-CLASS
     SUBCLASS OF top
     MAY CONTAIN {
          description,
          localityName,
          stateOrProvinceName,
          searchGuide,
          seeAlso,
          streetAddress}
     ::= {objectClass 3}

organization  OBJECT-CLASS
     SUBCLASS OF top
     MUST CONTAIN {
          organizationName}
     MAY CONTAIN {
          organizationalAttributeSet}
     ::= {objectClass 4}

organizationalUnit  OBJECT-CLASS
     SUBCLASS OF top
     MUST CONTAIN {
          organizationalUnitName}
     MAY CONTAIN {
          organizationalAttributeSet}
     ::= {objectClass 5}
```

```
person OBJECT-CLASS
      SUBCLASS OF top
      MUST CONTAIN {
            commonName,
            surname}
      MAY CONTAIN {
            description,
            seeAlso,
            telephoneNumber,
            userPassword}
      ::= {objectClass 6}

organizationalPerson OBJECT-CLASS
      SUBCLASS OF person
      MAY CONTAIN {
            localeAttributeSet,
            organizationalUnitName,
            postalAttributeSet,
            telecommunicationAttributeSet,
            title}
      ::= {objectClass 7}

organizationalRole OBJECT-CLASS
      SUBCLASS OF top
      MUST CONTAIN {
            commonName}
      MAY CONTAIN {
            description,
            localeAttributeSet,
            organizationalUnitName,
            postalAttributeSet,
            preferredDeliveryMethod,
            roleOccupant,
            seeAlso,
            telecommunicationAttributeSet}
      ::= {objectClass 8}

groupOfNames OBJECT-CLASS
      SUBCLASS OF top
      MUST CONTAIN {
            commonName,
            member}
      MAY CONTAIN {
            description,
            organizationName,
            organizationalUnitName,
            owner,
            seeAlso,
            businessCategory}
      ::= {objectClass 9}

residentialPerson OBJECT-CLASS
      SUBCLASS OF person
      MUST CONTAIN{
            localityName}
      MAY CONTAIN {
            localeAttributeSet,
            postalAttributeSet,
            preferredDeliveryMethod,
            telecommunicationAttributeSet,
            businessCategory}
      ::= {objectClass 10}
```

```
applicationProcess  OBJECT-CLASS
     SUBCLASS OF top
     MUST CONTAIN {
          commonName}
     MAY CONTAIN {
          description,
          localityName,
          organizationalUnitName,
          seeAlso}
     ::= {objectClass 11}

applicationEntity  OBJECT-CLASS
     SUBCLASS OF top
     MUST CONTAIN {
          commonName,
          presentationAddress}
     MAY CONTAIN {
          description,
          localityName,
          organizationName,
          organizationalUnitName,
          seeAlso,
          supportedApplicationContext}
     ::= {objectClass 12}

dSA  OBJECT-CLASS
     SUBCLASS OF applicationEntity
     MAY CONTAIN {
          knowledgeInformation}
     ::= {objectClass 13}

device  OBJECT-CLASS
     SUBCLASS OF top
     MUST CONTAIN {
          commonName}
     MAY CONTAIN {
          description,
          localityName,
          organizationName,
          organizationalUnitName,
          owner,
          seeAlso,
          serialNumber}
     ::= {objectClass 14}

strongAuthenticationUser  OBJECT-CLASS
     SUBCLASS OF top
     MUST CONTAIN { userCertificate }
     ::= {objectClass 15}

certificationAuthority  OBJECT-CLASS
     SUBCLASS OF top
     MUST CONTAIN {
          cACertificate,
          certificateRevocationList,
          authorityRevocationList  }
     MAY CONTAIN { crossCertificatePair}
     ::= {objectClass 16}

END
```